区块链技术与应用

北京大学

肖臻 研究员.

同学们受苦了
老师讲的好，都是关键概念，很精炼



区块链技术与应用

第2讲：比特币中的密码学原理

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

比特币本质：crypto-currency.

比特币用到密码学中的2个功能. →
- ① 哈希 cryptographic hash function.
- ② 签名.（非对称加密）

一、哈希

cryptographic hash function 有3个重要性质
- ① collision resistance
- ② hiding
- ③ puzzle friendly

① collision resistance（抗碰撞性）

其中的 collision 是指"哈希碰撞" 即 输入 $X \neq Y$，也可以使得 $H(X)=H(Y)$ 也即"不同输入映射到哈希表中同一个位置"。

而事实上是，这种 collision 哈希碰撞很难被人工创造。给定 m，可求得 $H(m)$，但很难找到 m'，使 $H(m')=H(m)$.

这个特性的实践

上传一个文件到 cloud，如何防止该文件被篡改？用这个文件作为一个 input，计算这个 input 的哈希值 H(input) 存在本地，并把文件上传，之后将 cloud 上的文件下载后再计算其哈希值，比较两个哈希值是否一致，就可知文件是否被篡改。

MD5，一个很流行的哈希函数，但目前已经知道如何去制造 MD5 的哈希碰撞

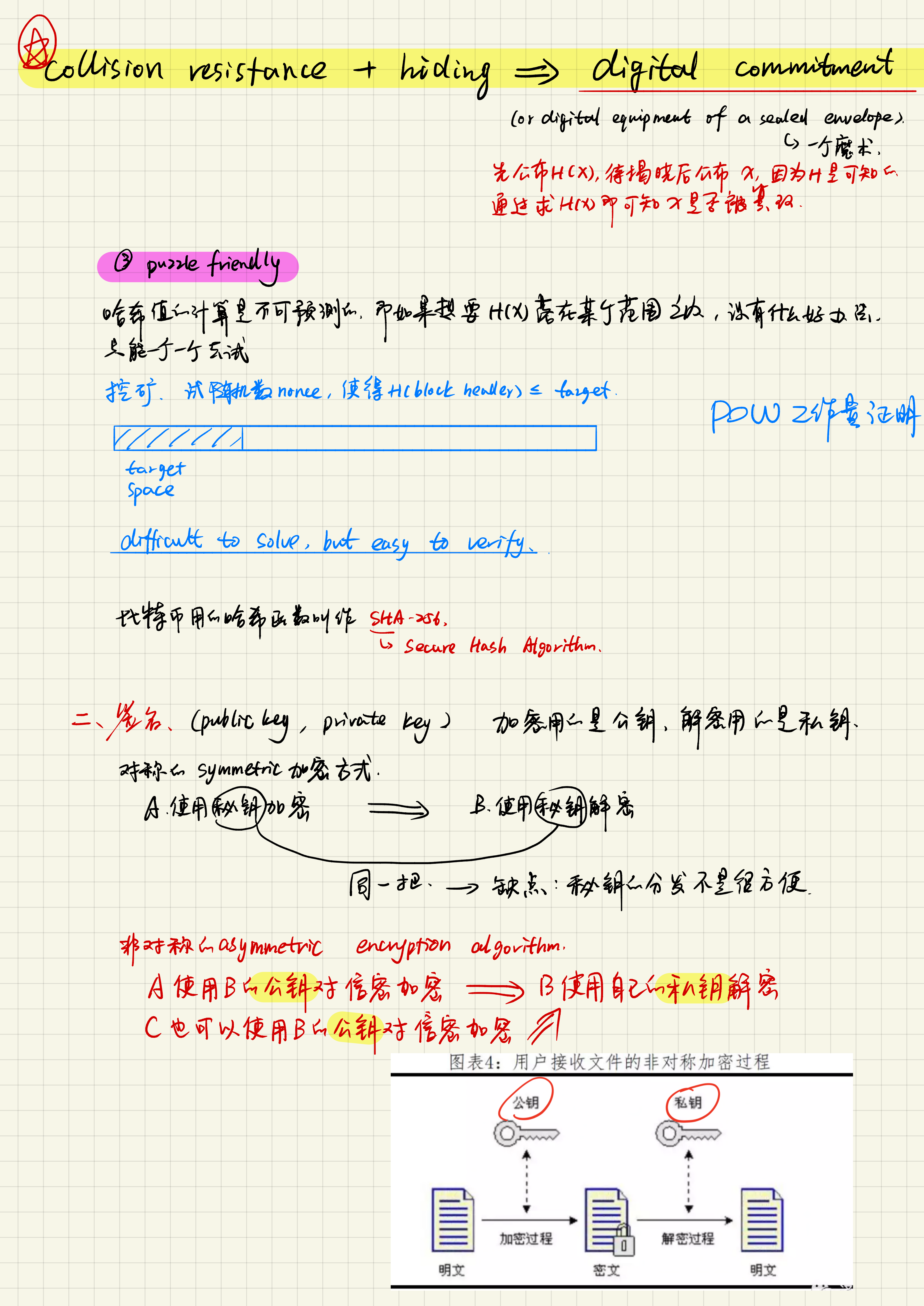
② hiding（单向不可逆性）

$X \rightarrow H(X)$

$H(X) \nrightarrow X$

⟺ 即，给定一个 X，可计算得 $H(X)$，但给定一个 $H(X)$，无法计算得 X.（除非暴力破解）

条件：
- 输入空间足够大
- 分布足够均匀



collision resistance + hiding ⇒ digital commitment

(or digital equipment of a sealed envelope)

↳ 一个魔术.

先公布H(X)，待揭晓后公布 x，因为H是可知的，通过求H(x)即可知x是否被篡改.

③ puzzle friendly

哈希值的计算是不可预测的. 即如果想要H(X)落在某个范围之内，没有什么好办法，只能一个一个去试

挖矿. 试随机数 nonce，使得 H(block header) ≤ target.

POW 工作量证明

target space

difficult to solve, but easy to verify.

比特币用的哈希函数叫作 SHA-256，

↳ Secure Hash Algorithm.

二、签名. (public key, private key)

加密用的是公钥，解密用的是私钥.

对称的 symmetric 加密方式.

A.使用秘钥加密 ⇒ B.使用秘钥解密

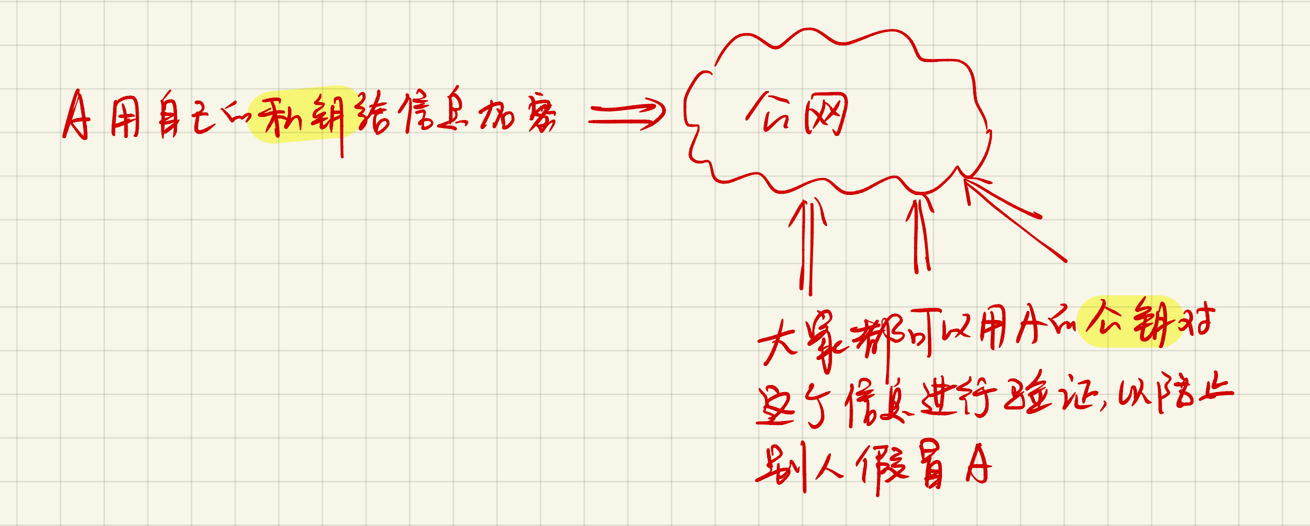
同一把. → 缺点：秘钥的分发不是很方便.

非对称的 asymmetric encryption algorithm.

A 使用B的公钥对信息加密 ⇒ B使用自己的私钥解密

C 也可以使用B的公钥对信息加密

图表4：用户接收文件的非对称加密过程



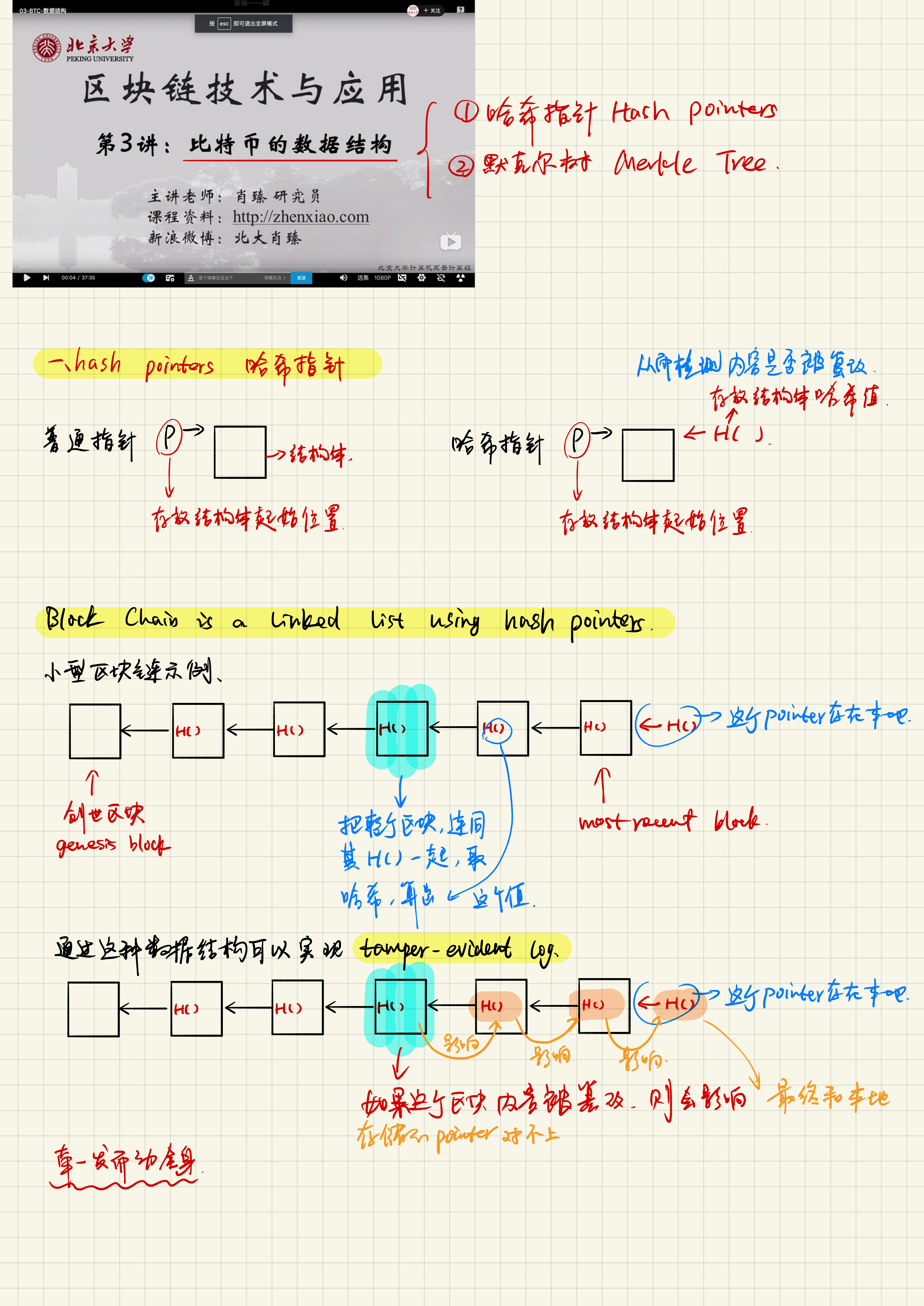
在比特币区块链中，私钥代表了对比特币的控制权。交易发起方用私钥对交易（包括转账金额和转账地址）签名并将签名后的交易和公钥广播，各节点接收到交易后可以用公钥验证交易是否合法。在这个过程中交易发起方无须暴露自己的私钥，从而实现保密目的。

比特币交易的实现：签名。

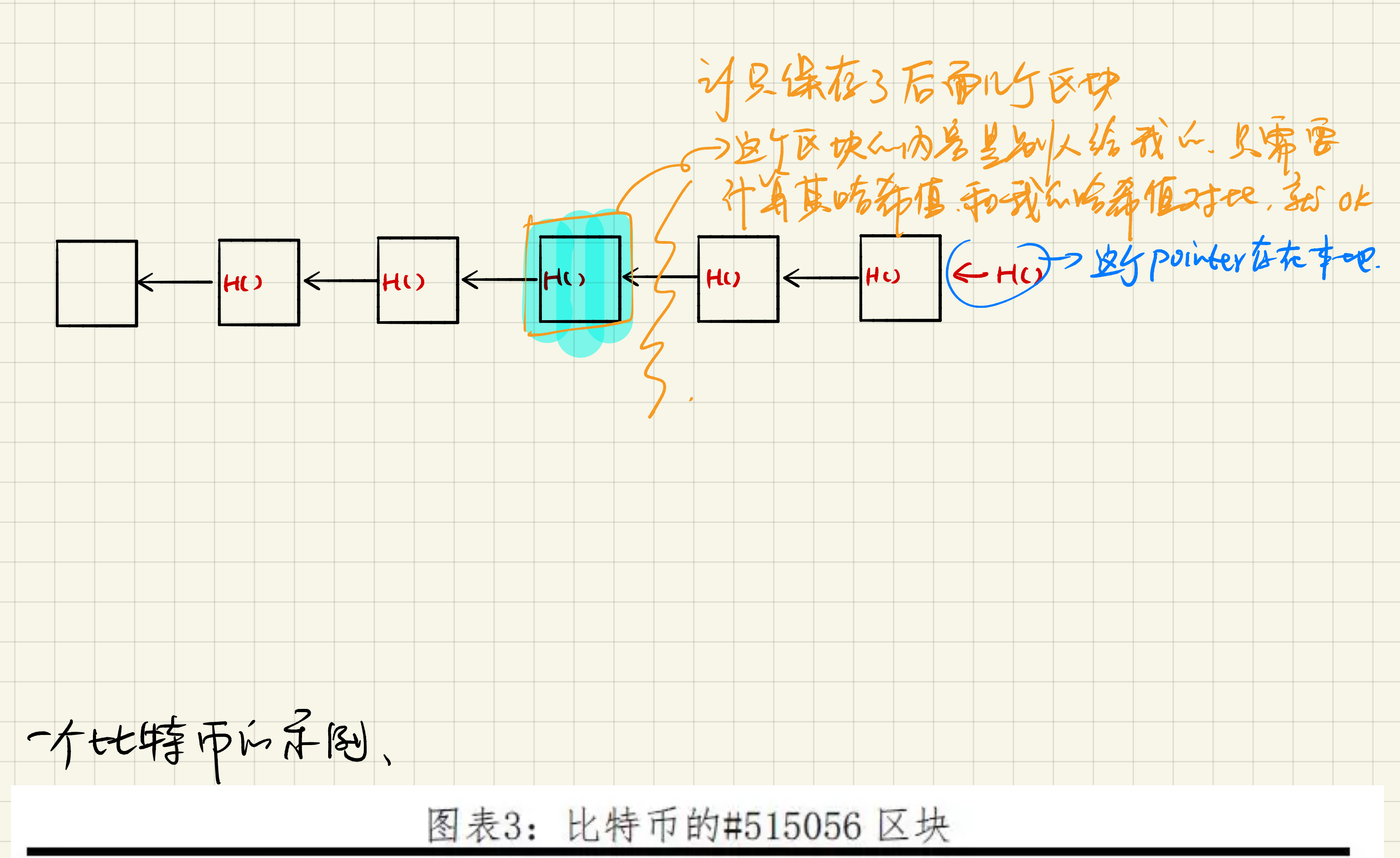
A要转10个比特币给B，A将转账信息发布在公网上，如何实现？

✓

✓



03-BTC-数据结构
北京大学
PEKING UNIVERSITY
区块链技术与应用
第3讲：比特币的数据结构
主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻
①哈希指针 Hash pointers
②默克尔树 Merkle Tree.
一、hash pointers 哈希指针
普通指针
P
结构体.
存放结构体起始位置.
哈希指针
P
H( ).
存放结构体哈希值.
从而检测内容是否被篡改.
存放结构体起始位置.
Block Chain is a linked list using hash pointers.
小型区块链示例、
H( )
创世区块
genesis block
把这个区块,连同其 H( ) 一起,取哈希,算出这个值.
most recent block.
这个pointer存在本地.
通过这种数据结构可以实现 tamper-evident log.
影响
影响
影响.
如果这个区块内容被篡改,则会影响最终和本地存储的 pointer 对不上
牵一发而动全身.



一个比特币的示图、

图表3：比特币的#515056 区块

Block #515056

BlockHash 0000000000000000001a6f7fc40321da61aada55d65aa199008737de24c99471 ← 头哈希

Summary

Number Of Transactions	182	Difficulty	3462542391191.563
Height	515056 (Mainchain)	Bits	17514a49
Block Reward	12.5 BTC	Size (bytes)	121831
Timestamp	Mar 25, 2018 1:18:55 PM	Version	536870912
Mined by		Nonce 随机数	526738450
Merkle Root (Root Hash)	80e9c530e82b643a8e39b6d68...		
Previous Block	515055 ← 父哈希		

块头

Transactions

ac8fb7336999c258f367fceeb6a62a4558e7162cfdb5f350865749310d969bc0 — mined Mar 25, 2018 1:18:55 PM

No Inputs (Newly Generated Coins) → 1C1mCxRukix1KfegAY5zQQJV7samAci2pv 12.51404902 BTC (U); Unparsed address [0] 0 BTC (U)

1 CONFIRMATIONS | 12.51404902 BTC

53b019ac43d31937255f2a3901a01c590671944197a25df671437325bd817117 — mined Mar 25, 2018 1:18:55 PM

32HhBWZE1mxikAH38ejPQHhZycmYiGiYwT 0.03488 BTC → 38PbQokXXubqQWCUuZPFkMPxADrLuMDzvK 0.03438 BTC (U)

FEE: 0.0005 BTC

1 CONFIRMATIONS | 0.03438 BTC

块身

资料来源：blockexplorer，恒大研究院

二、Merkle Tree

Merkle Tree vs. Binary Tree

Merkle Tree 示例.

H( ) 根哈希. (root hash)

H( ) H( ) → 根节点

同

两个哈希值拼在一起，再取个哈希存在上面这个节点

H( ) H( )　　H( ) H( )

H( ) H( )　H( ) H( )　H( ) H( )　H( ) H( )

哈希值存在上面这个节点

hash pointers

data blocks 数据块

数据块

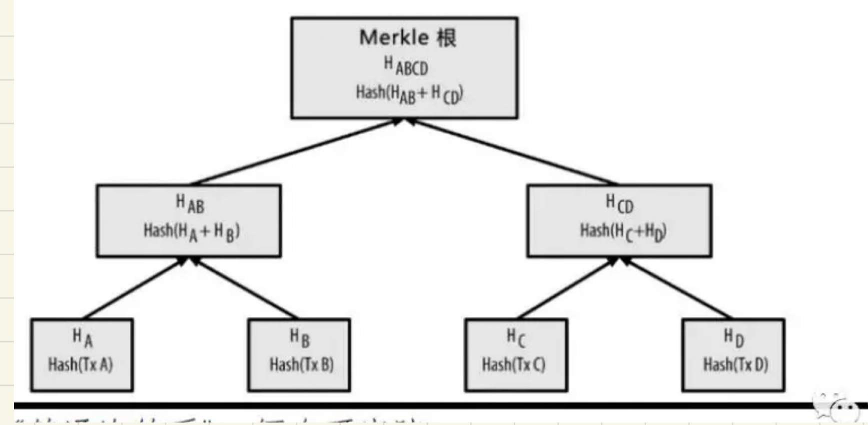
tx　tx　tx　tx　tx　tx　tx　tx

比特币中，每个数据块其实是一种交易 tx. (Transactions)

Merkle Tree 的好处

只要存该 root hash，就能检测出树中任意节点的修改.

✔



图表7：一棵包含了四个交易元素的默克尔树

每个区块分为2个部分 {
- Block header 块头. 有根哈希值, 没有交易具体内容
- Block body 块身. 有交易列表.

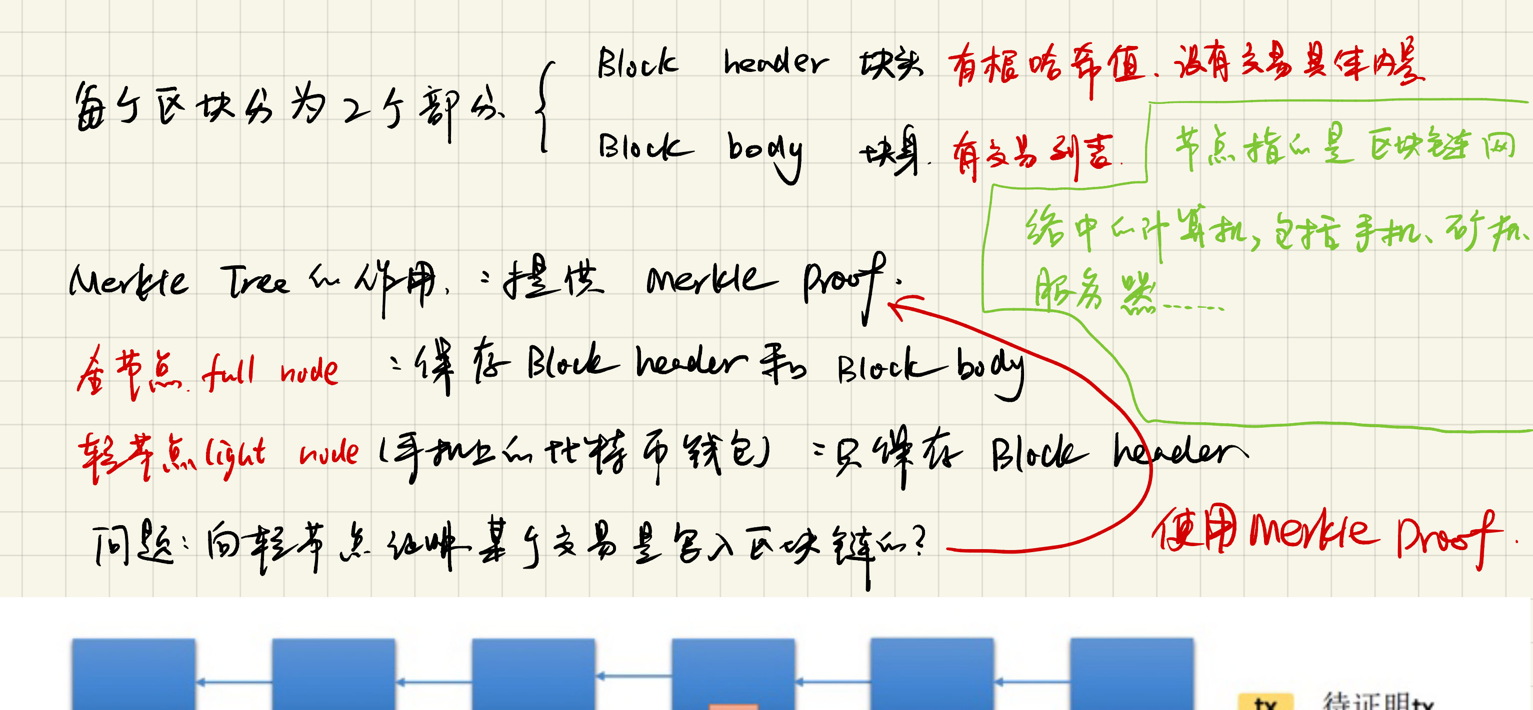
节点指的是区块链网络中的计算机，包括手机、矿机、服务器......

Merkle Tree 的作用：提供 merkle proof.

全节点 full node：保存 Block header 和 Block body

轻节点 light node (手机上的比特币钱包)：只保存 Block header

问题：向轻节点证明某个交易是否写入区块链的？ → 使用 Merkle Proof.



问题：① proof of membership / proof of inclusion 如何证明

即上. 复杂度为 $\Theta(\log(n))$ 假设这棵树有n个数据块

② proof of non-membership.

方法是遍历, 复杂度为 $\Theta(n)$

但是可以对叶节点按哈希值大小进行排序, 用二分法, 对2个相邻的数据块都分别向上取哈希, 直到 root hash 并验证. 复杂度为 $\Theta(\log(n))$

这种 Tree 称为 sorted Merkle Tree (比特币没有用)



区块链技术与应用

第4讲：比特币的共识协议

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

问题：央行CB如何发行数字货币？

方案1：asymmetric encryption algorithm.

100元 signed by CB.

支付即复制，是否可以 → 不可以，Double Spending Attack 双花攻击.

方案2：有一张表，记录每一张货币的主人，在交易前验证这张数字货币是否是支付人的

017 → 肖臻
018 → 张三
019 → 李四

→ 可以，但是这种方案是中心化的

→ 去中心化，即将验证的职责，从由央行承担，改为由大众承担

数字货币发行需要解决两个问题

① 谁有权力发行

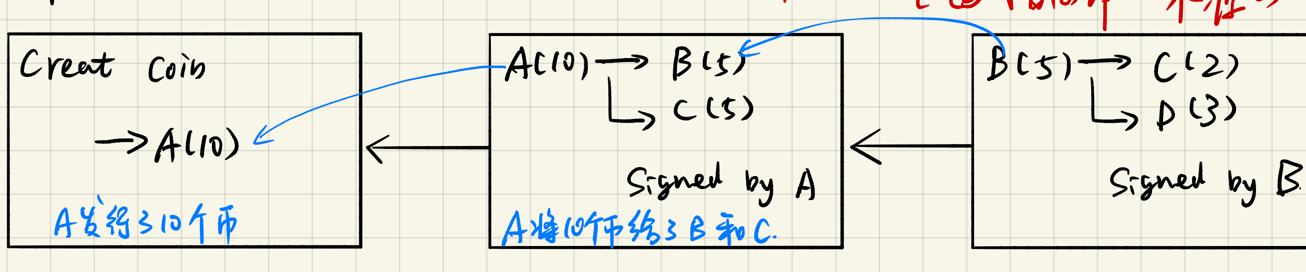
② 怎么验证交易有效性，防范双花攻击

一、怎么验证交易有效性

先导1：比特币如何进行交易

解决方案：Block Chain

有2类hash pointer { ① 块与块之间的 ② 指向币的来源的



数字货币交易包括 input 和 output.
- input：说明币的来源和支付人的公钥
- output：说明支付给别人(B&C)的公钥Hash.

几个问题

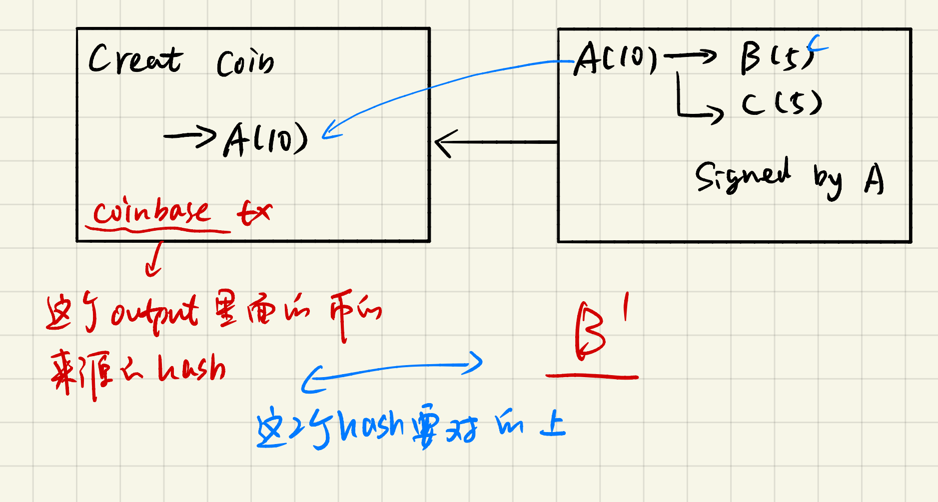
① 转账交易中，A（支付人）需要知道B（收款人）什么信息吗？

答：A需要知道B的地址，这个地址是由B的公钥经过一定计算得出的

② 转账交易中，B（收款人）需要知道A（支付人）什么信息吗？

答：B要知道A的公钥，即知道A的身份。
所有节点都需要知道A的公钥（用于验证）

③ 如果有个B'宣称自己是A，使用A的公钥发起交易，如何防治？



如何将交易信息写进区块链？

账本的内容要取得分布式的共识（distributed consensus）

case：分布式哈希表（distributed hash），即系统里许多节点共同维护

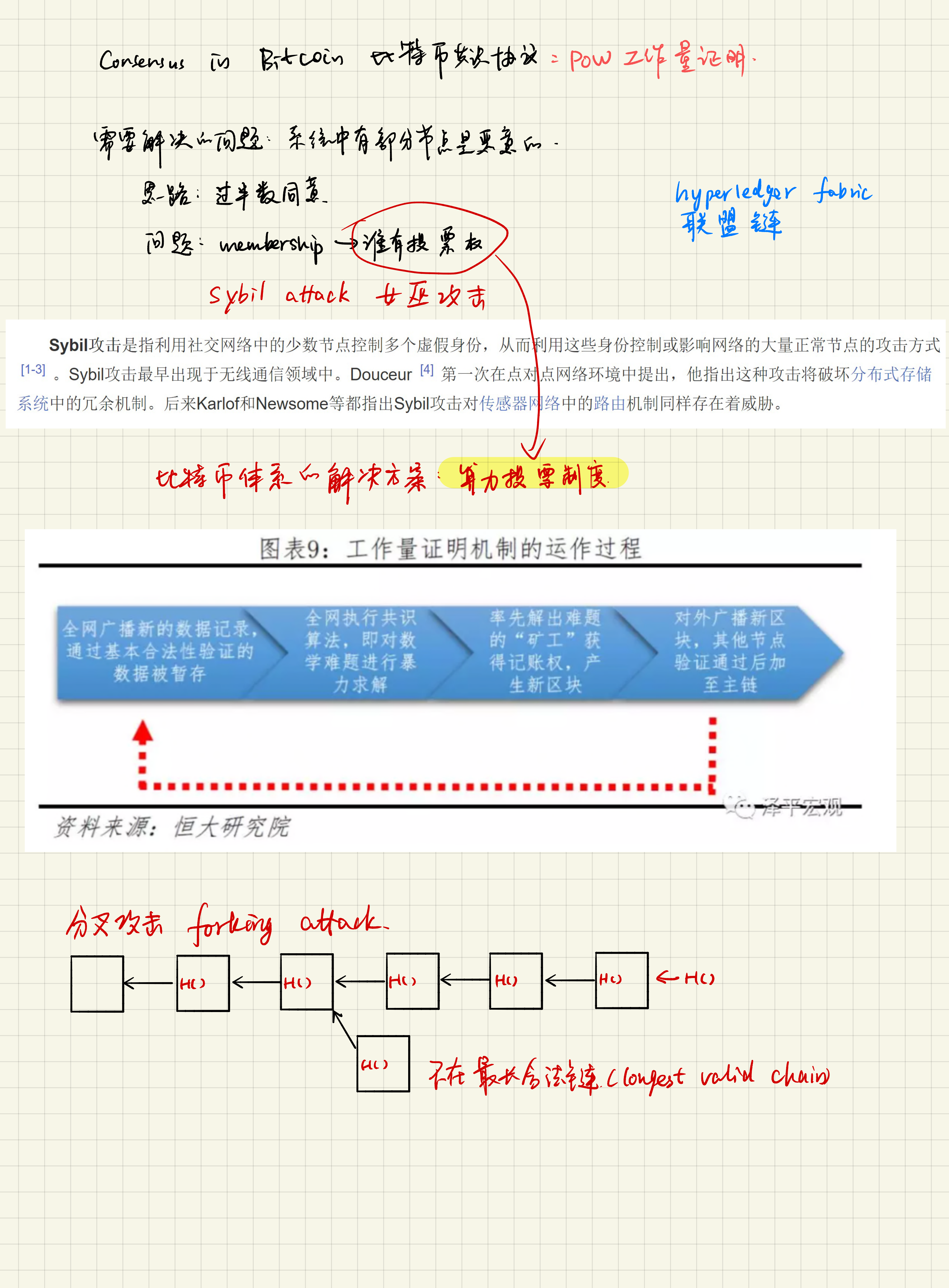
FLP impossibility result（不可能结论）：

在一个异步（asynchronous）系统里，网络传输迟延没有上限，哪怕系统中有一个成员是faulty，也无法达成共识

FLP给出了一个令人吃惊的结论：在异步通信场景，即使只有一个进程失败，也没有任何算法能保证非失败进程达到一致性！

CAP Theorem

CAP原则又称CAP定理，指的是在一个分布式系统中，一致性（Consistency）、可用性（Availability）、分区容错性（Partition tolerance）。CAP 原则指的是，这三个要素最多只能同时实现两点，不可能三者兼顾。



Consensus in Bitcoin 比特币共识协议 = PoW 工作量证明.
需要解决的问题：系统中有部分节点是恶意的.
思路：过半数同意.
问题：membership → 谁有投票权
Sybil attack 女巫攻击
hyperledger fabric
联盟链
Sybil攻击是指利用社交网络中的少数节点控制多个虚假身份，从而利用这些身份控制或影响网络的大量正常节点的攻击方式[1-3]。Sybil攻击最早出现于无线通信领域中。Douceur [4] 第一次在点对点网络环境中提出，他指出这种攻击将破坏分布式存储系统中的冗余机制。后来Karlof和Newsome等都指出Sybil攻击对传感器网络中的路由机制同样存在着威胁。
比特币体系的解决方案：算力投票制度.
图表9：工作量证明机制的运作过程
全网广播新的数据记录，通过基本合法性验证的数据被暂存
全网执行共识算法，即对数学难题进行暴力求解
率先解出难题的“矿工”获得记账权，产生新区块
对外广播新区块，其他节点验证通过后加至主链
泽平宏观
资料来源：恒大研究院
分叉攻击 forking attack
H()
H()
H()
H()
H()
← H()
H()
不在最长合法链 (longest valid chain)

同时挖到矿，出现2个等长区块 接受哪个？

维持一段时间，看哪个区块先被接上

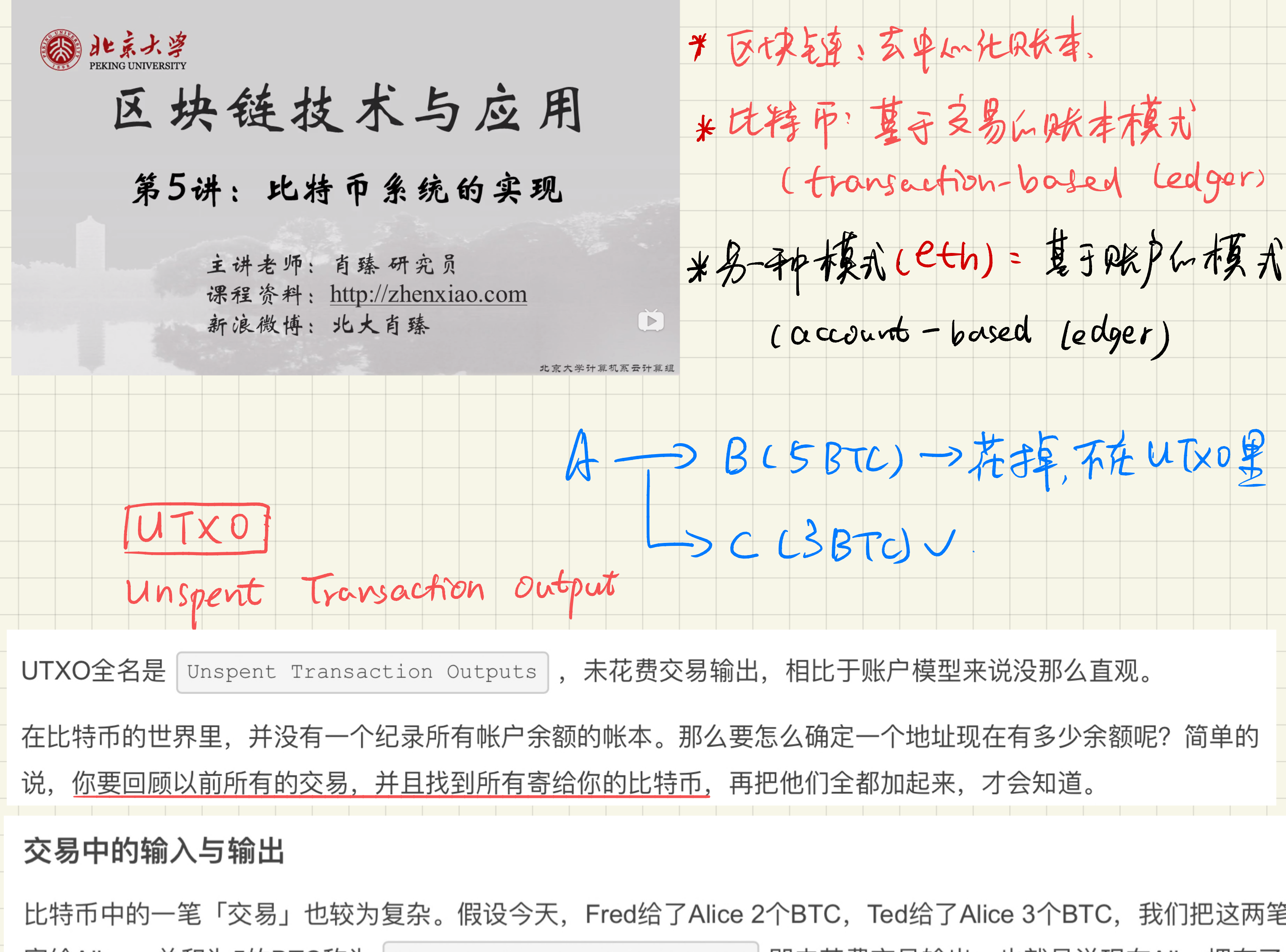
二、为什么要挖？ mining 挖矿

Block reward 出块奖励. coinbase tx 唯一一个产生新币的途径

50BTC → 25BTC → 12.5BTC → ···

每21万个区块减少1半

目前的情况



UTXO全名是 `Unspent Transaction Outputs` ，未花费交易输出，相比于账户模型来说没那么直观。

在比特币的世界里，并没有一个纪录所有帐户余额的帐本。那么要怎么确定一个地址现在有多少余额呢？简单的说，你要回顾以前所有的交易，并且找到所有寄给你的比特币，再把他们全都加起来，才会知道。

交易中的输入与输出

比特币中的一笔「交易」也较为复杂。假设今天，Fred给了Alice 2个BTC，Ted给了Alice 3个BTC，我们把这两笔寄给Alice，总和为5的BTC称为 `Unspent Transaction Outputs` 即未花费交易输出：也就是说现在Alice拥有了两笔Unspent Transcation Outputs，可以当作他未来转钱给别人的 `input` 。

如果现在Alice想要转5 BTC给Bob，他要将前面两笔总和刚好为5的UTXO当作这笔交易的输入。而矿工要验证的就是并没有其他交易在先前的区块当中，已经使用过这笔 `Unspent Output` 。如果同一笔输出已经被发送过，那它就不是 `Unspent` 了，这就是比特币预防 `Double Spending` 的方法。

全节点维护UTXO，以便快速检测Double Spending

现实中的比特币交易：

3a9e71c7029cc806f3de37dd66b2ab49fbd2765878323f4db3fd427895e6cf2b — mined Jan 31, 2018 2:03:07 PM

Input		Output	
1KjyKwSVuGtqWW1kzKxR5vPeA8HG9yaC7B	56.61735492 BTC	1KvgPGFkJWHiNgejXH62UvpC9Z8Yg3JZUJ	0.03 BTC (S)
		1BuVvSEhwFZuidceKdciA41KbAGpChNvM4	0.2 BTC (S)
		1HpQCA4h9G1NPhEPkxcjYyDmRjigK8Wgbe	56.38706253 BTC (S)

FEE: 0.00029239 BTC — 1579 CONFIRMATIONS — 56.61706253 BTC

UTXO.

给节点的交易费

可能转给自己

total input (左) = total output (右)

比特币的具体示例、

Block Example

Block #529709 区块

Summary	
Number Of Transactions 共有686笔交易	686
Output Total	4,220.46616378 BTC
Estimated Transaction Volume	651.93844862 BTC
Transaction Fees 总交易费	0.12458867 BTC
Height	529709 (Main Chain)
Timestamp	2018-06-29 06:17:26
Received Time	2018-06-29 06:17:26
Relayed By	BTC.com
Difficulty	5,077,499,034,879.02
Bits	389508950
Size	333.53 kB
Weight	1160.618 kWU
Version	0x20000000
Nonce 最后找到的符合要求的随机数	3897564446
Block Reward 出块奖励	12.5 BTC



Hashes 块头hash值.	
Hash	000000000000000000030f1531dbfa069037188c5048b23f0cb979ce8728ed8c5
Previous Block	00000000000000000000841c59a4679d6e707152f24f5b195b66b47397d65175e
Next Block(s) 前一个区块的块头hash值.	
Merkle Root	6f73d36264f05e0c55c4703f85e85e628a29231e9673bcef3c02bfe76dc2b378



source: blockchain.info

比特币交易的实例

Transaction View information about a bitcoin transaction

3fd0d94dcf733a614f14a930a470241e0d99ea6966f999f1fa6f895396a6645f

14BHEP2sNRUJj5jSexgBjyszza87psR2Uv (0.00408688 BTC - Output)
1K7q5sqzbTTttdd6MtZDHb9E4jjCShiUcR (0.04758063 BTC - Output)

→

1Kqjq3Qiqbd1ah9G9imm8comNDKgxVvLVF - (Unspent) 0.0396 BTC
17Eu6pyMUJZHoaEKq3u8k2D3izdw9QYRT - (Unspent) 0.01019031 BTC

1 Confirmations | 0.04979031 BTC

Summary	
Size	373 (bytes)
Weight	1492
Received Time	2018-06-29 06:13:26
Lock Time	Block: 529708
Included In Blocks	529709 (2018-06-29 06:17:26 + 4 minutes)
Confirmations	1 Confirmations
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.05166751 BTC
Total Output	0.04979031 BTC
Fees	0.0018772 BTC
Fee per byte	503.271 sat/B
Fee per weight unit	125.818 sat/WU
Estimated BTC Transacted	0.0396 BTC
Scripts	Hide scripts & coinbase

Input Scripts 输入脚本、将之与币来源的输入脚本验证

ScriptSig: PUSHDATA(71)
[304402200e902feaaf8e49ea467bbc3d9034bdf33fedfbfbb662e75e8822372a3c0871e102204176d40c1781ca6f465d47db3628e2610f53d5a54ceb24e6118296ae71df5e5201]
PUSHDATA(33)[03b339dc9b56131ad95753f6995808fcc2c686bad4f69ea0b9bc9e564315c1e515]

ScriptSig: PUSHDATA(72)
[3045022100eade6baa42d209d279033581a593340780181aa0dd8a7fd2d5b6162acd81ca4d022074bd1a62c6138505823efae9ec62d4dd16e57c454a245be25f961a6e8144930201]
PUSHDATA(33)[02ceddac2ca907c010dfea74dc3c4bc27a17dcab0a2e88993cdccc243c818279ed]

Output Scripts

DUP HASH160 PUSHDATA(20)[cea9fb4638839ad9ebc9c8382dd03e2666aaeb65] EQUALVERIFY CHECKSIG

DUP HASH160 PUSHDATA(20)[4471a74ad7287cbbf6e6d1c092b22b55c886c1de] EQUALVERIFY CHECKSIG

source: blockchain.info

挖矿过程的解释：

每次挖矿过程就是一次 Bernoulli trail：a random experiment with binary outcome

每次试 nonce 求解，构成了 Bernoulli process：（无记忆性）a sequence of independent Bernoulli trail.

实验次数很多，每次实验的成功率很低、⟹ Poisson process

出块时间服从指数分布。

probability density（概率密度）

average.

10 mins.

time to block（出块时间）

progress free：

无论是指数分布，还是 Bernoulli process，都是 memoryless，或称 progress free，即从中途任意一点开始，其成功概率不变，这种性质给予算力成比例优势。

比特币总量。　　挖矿并不是解数学题，挖矿难度是人为设定的

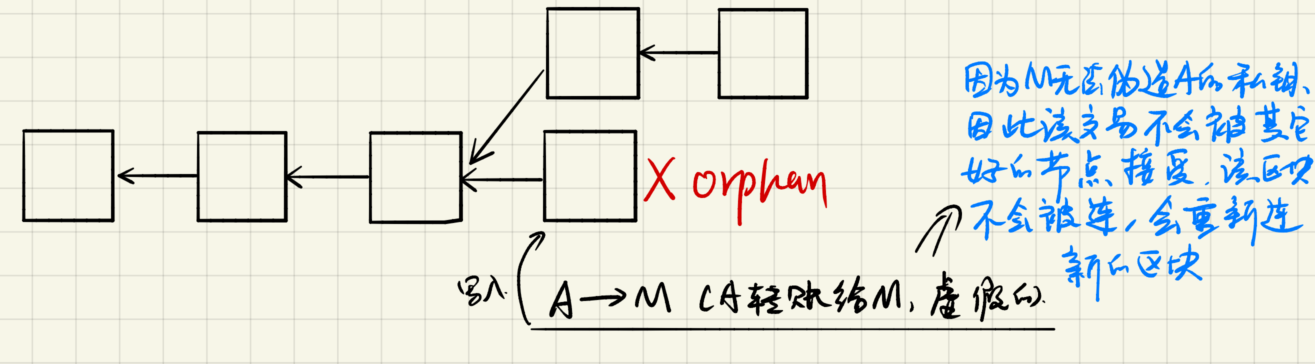
$$21万 \times 50\,BTC + 21万 \times 25\,BTC + 21万 \times 12.5\,BTC + \cdots$$
$$= 21万 \times 50BTC \times (1 + \frac{1}{2} \times \frac{1}{4} + \cdots)$$
$$= 21万 \times 50 \times 2 = 2100万个$$

geometric series

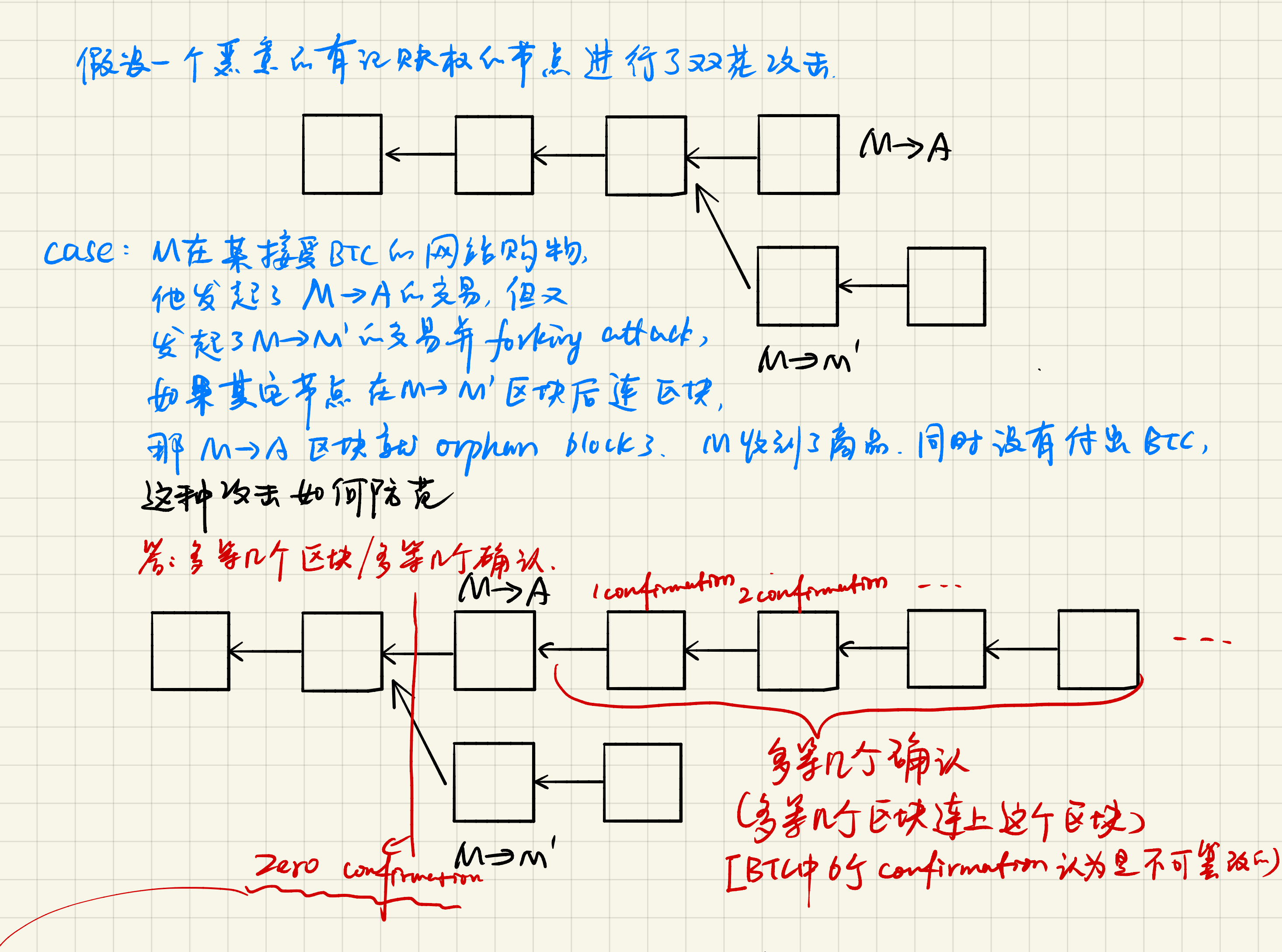
Bitcoin is secured by mining

问题：假设大部分算力掌握在诚实的节点手里，可否保证所有写入区块的交易都是合法的？

假设一个恶意的有记账权的节点写入一个伪造的交易



假设一个恶意的有记账权的节点进行了双花攻击.



case: M在某接受BTC的网站购物, 他发起了M→A的交易, 但又发起了M→M'的交易并forking attack, 如果其它节点在M→M'区块后连区块, 那M→A区块就orphan block了. M收到了商品. 同时没有付出BTC,

这种攻击如何防范

答: 多等几个区块/多等几个确认.

区块链是irrevocable ledger、但这种不可篡改性只是概率上的保证, 刚刚写入Block的交易、是容易篡改的

关于zero confirmation.

① Block chain 节点缺省设置是先接受最先监听收到的交易并写入区块链.
② 电商购物有天然发货时间保护

假设一个恶意的有记账权的节点就是不把合法交易写进区块

答: 不碍事. 因为可以等到下一个区块.
事实上, BTC本来对区块中交易数量有大小限制. 最大不超过1M.
如果这个区块写不下, 就等下一个区块.

Selfish Mining: 自己藏一堆块不发布, 等攒一段时间后再一起发布.



区块链技术与应用

第6讲：比特币网络工作原理

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

Application layer (应用层)：运行 Bitcoin Block Chain

network layer (底层)：运行 P2P Overlay Network.

比特币网络设计原则：Simple, robust but not efficient.

flooding

每个节点都要维护一个等待上链的交易的集合.

该节点监听到了 A→B 的交易, 就将其写入集合

如果节点维护的等待上链的集合,

这时如果有一个 A→C 的双花攻击, 该节点是不会再写入的

如果该节点监听到区块中有同样一笔 A→B 的交易,
就会将集合中的该笔交易删除

如果该节点监听到区块中有一笔 A→C 的交易(用的同一币来源),
也会将集合中的 A→C 这笔交易删除

A→B → 写入
↓
A→C (非法) → 无法写入

题外话：1个区块大小 1M, 大概需要几十秒时间才能传到绝大多数的节点



区块链技术与应用

第7讲：比特币的挖矿难度调整

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

挖矿复习：

不断尝试nonce值，使Block header里的hash值，≤给定的目标阈值.

H(block header) ≤ target.　(target越小，挖矿难度越大)

比特币用的Hash算法是SHA256。这个hash值是256位的.

输出空间有 2^{256} 种可能的取值

$$difficulty = \frac{difficulty_1_target}{target}$$

→ difficulty_1_target：挖矿难度=1时所对应的目标阈值

为何要调整难度？出块时间太短会有什么问题？

分叉会容易出现，而且不止二分叉。分叉过多，对系统达成共识没有好处，且会危害系统的安全性

BTC出块速度是10分钟。ETH出块速度是15秒。意味着ETH需要新的协议(ghost.)

在ghost协议中，orphan block不能简单丢弃，而是要给奖励。uncle reward.

如何调整挖矿难度.

每2016个区块(约2周)调整一次目标阈值.

$$target = target \times \frac{actual\ time}{expected\ time}$$

target：越小越难.

actual time → 系统中产生2016个区块实际花费的时间

expected time → 产生2016个区块预计花费的时间(约14天)

如果有恶意的节点，不调整代码中的target，怎么办

如果不调，发布的区块，诚实的矿工不会认可。

在Block header中，有个nbits，这个nbits是target编码的版本

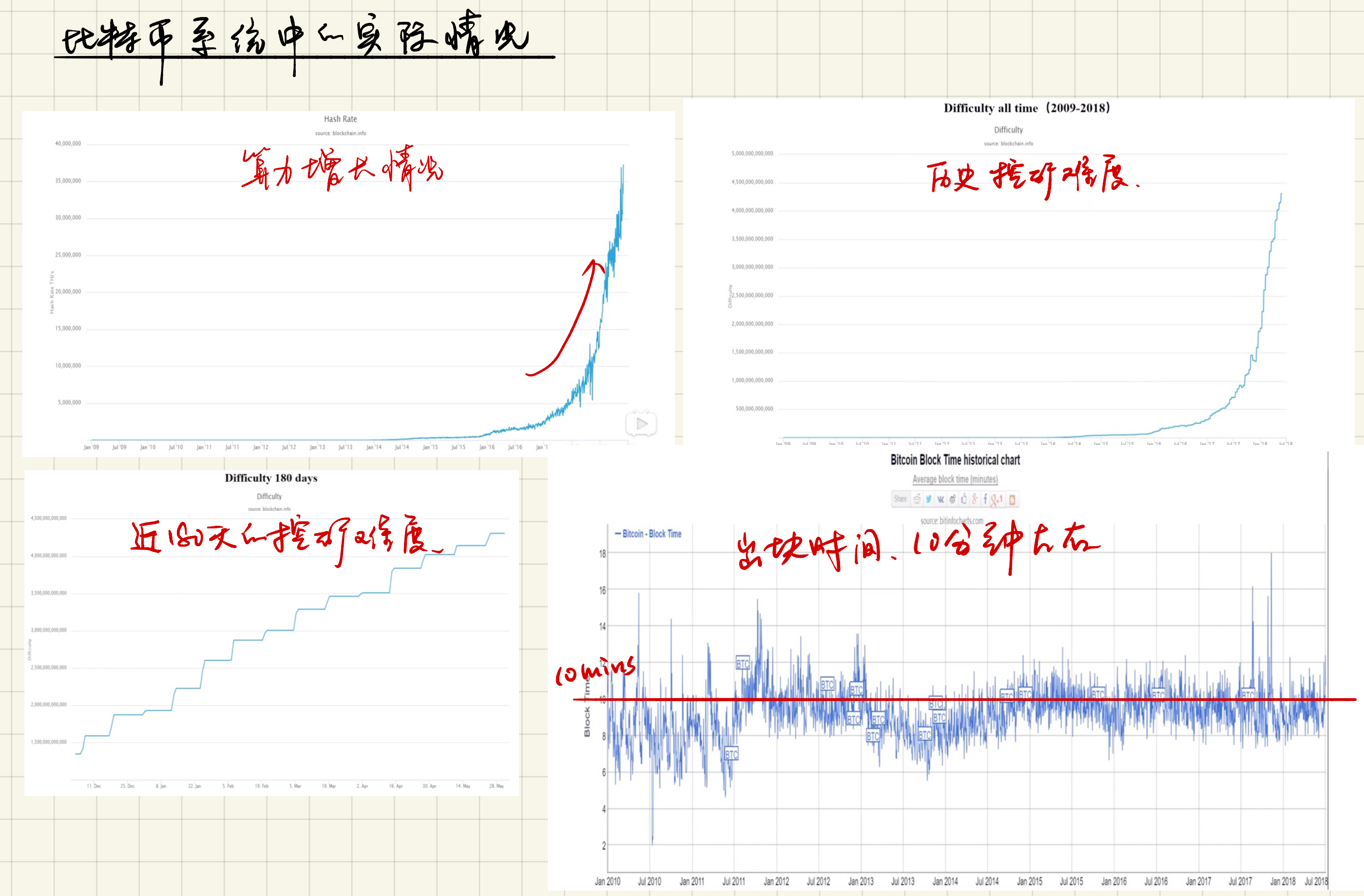
Block #529709

Summary	
Number Of Transactions	686
Output Total	4,220.46616378 BTC
Estimated Transaction Volume	651.93844862 BTC
Transaction Fees	0.12458867 BTC
Height	529709 (Main Chain)
Timestamp	2018-06-29 06:17:26
Received Time	2018-06-29 06:17:26
Relayed By	BTC.com
Difficulty	5,077,499,034,879.02
Bits	389508950
Size	333.53 kB
Weight	1160.618 kWU
Version	0x20000000
Nonce	3897564446
Block Reward	12.5 BTC

Hashes	
Hash	0000000000000000000030f1531dbfa069037188c5048b23f0cb979ce8728ed8c5
Previous Block	00000000000000000000841c59a4679d6e707152f24f5b195b66b47397d65175e
Next Block(s)	
Merkle Root	6f73d36264f05e0c55c4703f85e85e628a29231e9673bcef3c02bfe76dc2b378

source: blockchain.info

比特币系统中的实际情况





区块链技术与应用

第8讲：比特币挖矿

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

总结&复习

全节点（一般来说一直在线）

- 一直在线
- 在本地硬盘上维护完整的区块链信息
- 在内存里维护UTXO集合，以便快速检验交易的正确性
- 监听比特币网络上的交易信息，验证每个交易的合法性
- 决定哪些交易会被打包到区块里
- 监听别的矿工挖出来的区块，验证其合法性
  ① 区块中每个交易是否合法
  ② 区块是否符合要求，即H(—) ≤ target 是否符合，target是否正确，并调整挖矿难度
  ③ 是否在最长合法链上。
- 挖矿
  - 决定沿着哪条链挖下去？
  - 当出现等长的分叉的时候，选择哪一个分叉？

轻节点

- 不是一直在线
- 不用保存整个区块链，只要保存每个区块的块头
- 不用保存全部交易，只保存与自己相关的交易
- 无法检验大多数交易的合法性，只能检验与自己相关的那些交易的合法性。
- 无法检测网上发布的区块的正确性
- 可以验证挖矿的难度
- 只能检测哪个是最长链，不知道哪个是最长合法链

比特币安全性的保证
① 密码学：没有私钥就没有签名。
② 共识机制

挖矿设备

Generation 1 : CPU

Generation 2 : GPU ⇝ 主要用于通用并行计算

Generation 3 : ASIC
(Application Specific Indergrated Circuit)

大型矿池

pool manager → 负责全节点要做的事

miner miner miner miner → 计算hash值

miners之间如何分配收益. PoW

假设要找的nonce，使得其hash值前70位为0符合要求，矿主只要规定找到nonce使得其hash前60位为0即可获得1个share，最后真正挖到BTC了，按share来分享收益。（70, 60位均为假设） ⇝ 这个nonce只能作为PoW，对矿主要来一点用都没有

Pool Concentration in China

Before we get into the best mining pools to join, it's important to note that most mining pools are in China. Many only have Chinese websites and support. Mining centralization in China is one of Bitcoin's biggest issues at the moment.

There are about 20 major mining pools. Broken down by the percent of hash power controlled by a pool, and the location of that pool's company, we estimate that Chinese pools control ~81% of the network hash rate:

China - 81%
Czech Republic - 10%
Iceland - 2%
Japan - 2%
Georgia - 2%
Russia - 1%

https://www.buybitcoinworldwide.com/mining/pools/

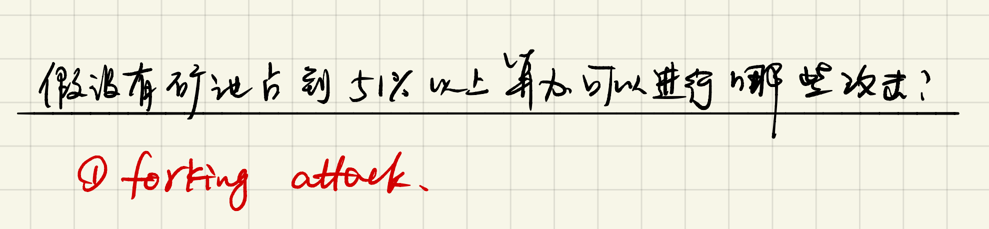
June 12, 2014
GHash.IO large mining pool crisis

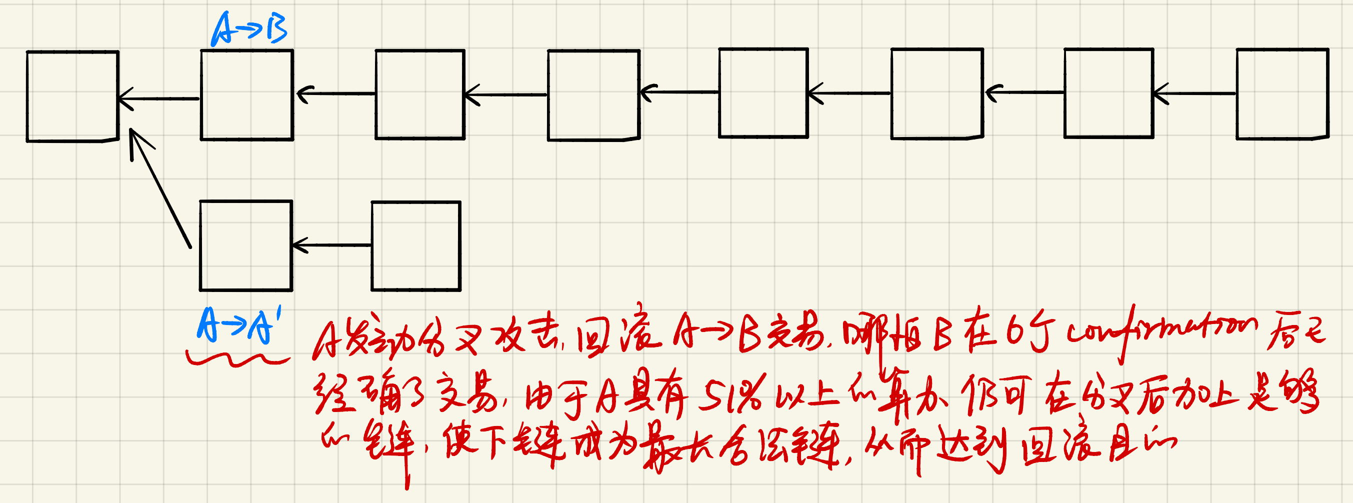
Other Known
Unknown
BitMinter
Slush
Polmine
Eligius
BTC Guild
Discus Fish
GHash.IO : 51 %
GHash.IO

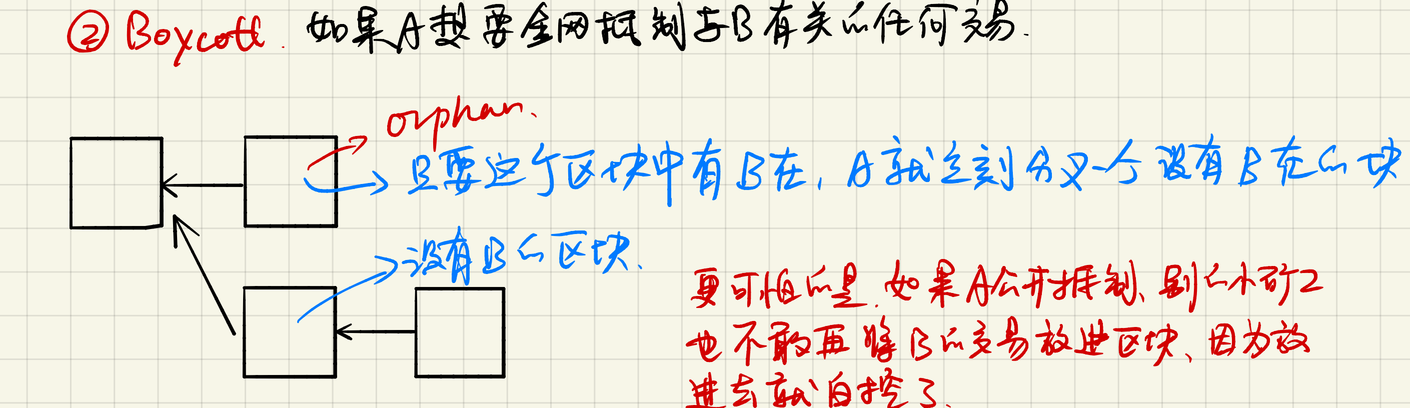
矿池算力分布

GHash.IO → 一度超过50% 有能力发动51% attack.

随后其自行减少算力


假设有矿池占到51%以上算力可以进行哪些攻击？
① forking attack.


A→B
A→A'
A发动分叉攻击，回滚A→B交易，哪怕B在6个confirmation后已经确认了交易，由于A具有51%以上的算力，仍可在分叉后加上足够的链，使下链成为最长合法链，从而达到回滚目的


② Boycott. 如果A想要全网抵制与B有关的任何交易.
orphan.
只要这个区块中有B在，A就立刻分叉一个没有B在的块
没有B的区块.
更可怕的是，如果A公开抵制，别的矿工也不敢再将B的交易放进区块，因为放进去就白挖了.

③ 盗币 → 是不可能的.



北京大学
PEKING UNIVERSITY

区块链技术与应用

第9讲：比特币脚本

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

北京大学计算机系云计算组

具体脚本如何实现见PPT（课件）.

交易实例.

Transaction View information about a bitcoin transaction

921af728159e3019c18bbe0de9c70aa563ad27f3f562294d993a208d4fcfdd24

1MaBFqBEfcQyXPv3fm5WAW9aQuJuKHaA3A (0.76469684 BTC - Output) ➡ 19z8LJkNXLrTv2QK5jgTncJCGUEEfpQvSr - (Unspent) 0.22684 BTC
1LvGTpdyeVLcLCDK2m9f7Pbh7zwhs7NYhX - (Spent) 0.53756644 BTC

23 Confirmations 0.76440644 BTC

23个确认，但滚可能性小

Summary	
Size	226 (bytes)
Weight	904
Received Time	2018-07-06 03:08:26
Included In Blocks	530657 (2018-07-06 03:12:07 + 4 minutes)
Confirmations	23 Confirmations
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.76469684 BTC
Total Output	0.76440644 BTC
Fees	0.0002904 BTC
Fee per byte	128.496 sat/B
Fee per weight unit	32.124 sat/WU
Estimated BTC Transacted	0.22684 BTC
Scripts	Hide scripts & coinbase

Input Scripts 基于栈

ScriptSig: PUSHDATA(72)
[3045022100928496fb0d2a25e4e7c99b9c60d4d0d12fcf8974a0faffcb30119b0d385872a30220253d3d0c507e5e44e123bc28b795ab4a38bf3b205455403e77aa72d58d9e17
PUSHDATA(33)[022ef8d3a6dd8a7039e513acc8ecf9b094ed7e85439824a1d11920f85927cd0018]

Output Scripts

DUP HASH160 PUSHDATA(20)[628ed6567c0b9056067309f07bbea2992ecad743] EQUALVERIFY CHECKSIG

DUP HASH160 PUSHDATA(20)[da7d57dfd02c6f5a9c649e891b5ac199ad012cd2] EQUALVERIFY CHECKSIG

用很少的fee将币、换取往这个区块里写入信息的机会

Proof of Burn

这个应用很重要 → digital commitment or sealed envelop.

缺点：发布交易不需要有记账权。
发布区块才需要有记账权。
因此，coinbase那个方法有局限性

output script:

```
RETURN
…[zero or more ops or text]
```

包含了这样的output script的output被称为Provably Unspendable/Prunable Outputs。

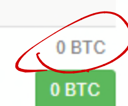
假如有一个交易的input指向这个output，不论input里的input script如何设计，执行到RETURN这个命令之后都会直接返回false，RETURN后面的其他指令也就不会执行了，所以这个output无法再被花出去，对应的UTXO也就可以被剪枝了，无需保存。

proof of burn 实例

Transaction View information about a bitcoin transaction

1a2e22a717d626fc5db363582007c46924ae6b28319f07cb1b907776bd8293fc

1MQaYLejR39TvN9PTxpAQcLBxFUqNHXx3M (0.05 BTC - Output) ➡ Unable to decode output address - (Unspent) 0 BTC



0 BTC

都是交易费

Summary	
Size	188 (bytes)
Weight	752
Received Time	2013-03-29 04:32:21
Included In Blocks	228596 (2013-03-29 14:18:58 + 587 minutes)
Confirmations	303536 Confirmations
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.05 BTC
Total Output	0 BTC
Fees	0.05 BTC
Fee per byte	26,595.745 sat/B
Fee per weight unit	6,648.936 sat/WU
Estimated BTC Transacted	0 BTC
Scripts	Hide scripts & coinbase

Input Scripts

ScriptSig: PUSHDATA(71)
[3044022055bcb36c829a614451787fe8c9bfb3798b683809b65b92037a015eccb5ff659702202461d2c708a4fd57c839e43634e8c02935d7b7d1db5b978432b0674c44645ec
PUSHDATA(33)[032c1ea520c25c4e66831cd395a3cd26f0e0a1472a3103fc8a4a63ef10e92d123c]

Output Scripts

RETURN PUSHDATA(20)[215477656e74792062797465206469676573742e]
(decoded) !Twenty byte digest.

写入的东西



北京大学
PEKING UNIVERSITY

区块链技术与应用

第10讲：比特币分叉 fork.

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

fork

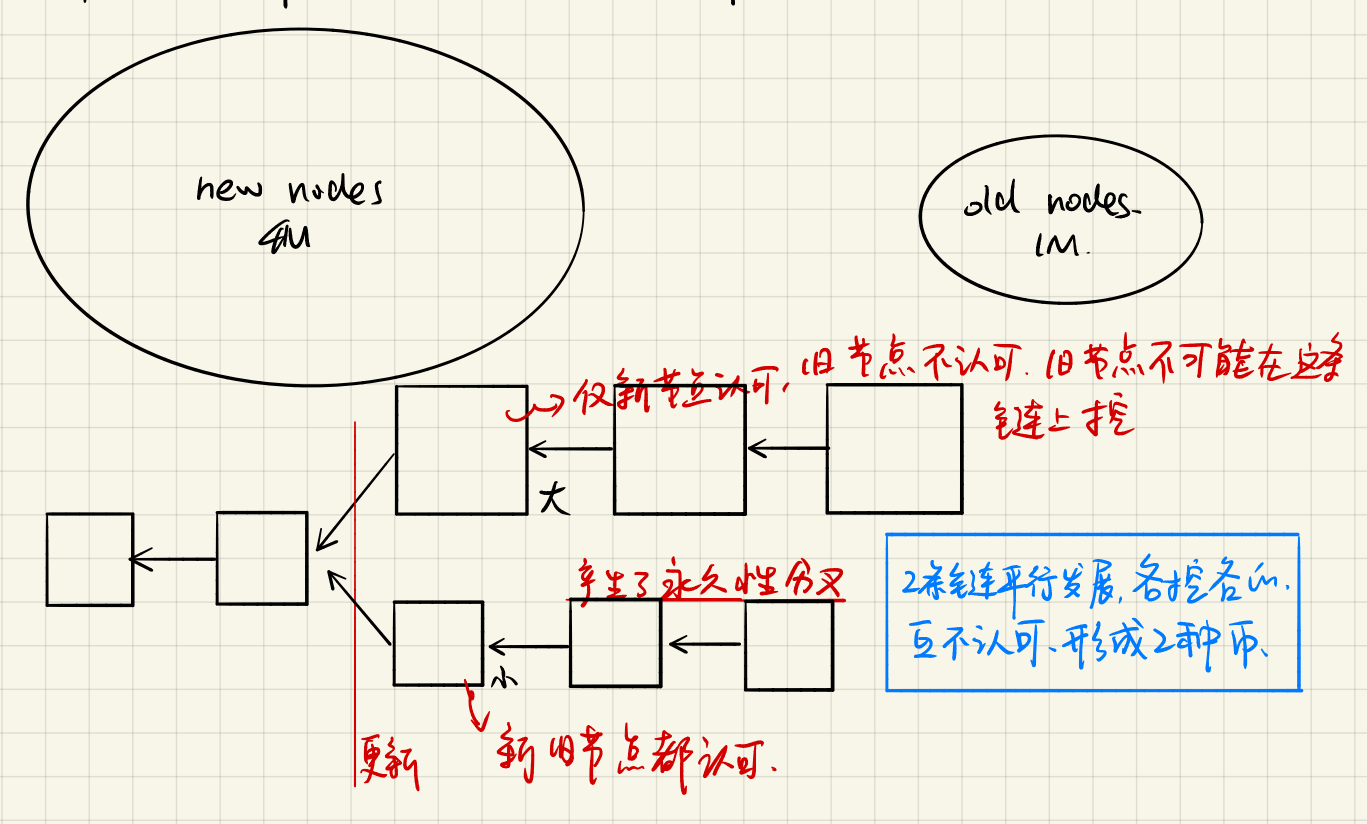
- state fork.
  - forking attack (deliberate fork).
- protocol fork (因为对BTC协议产生分歧而造成的分叉)
  - hard fork
  - soft fork

硬分叉. hard fork

block size limit

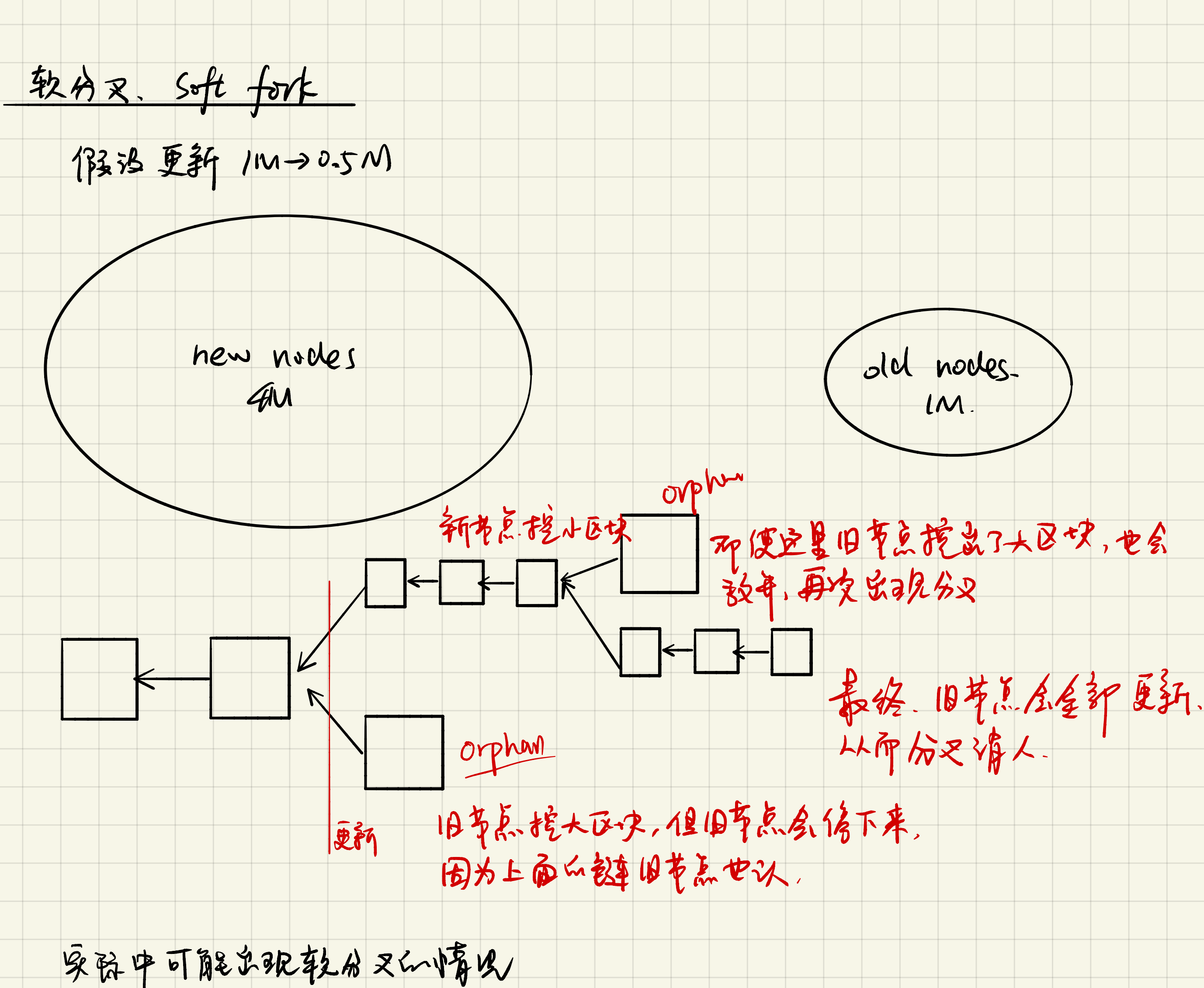
假设一次更新，1M→4M，大多数算力节点完成了更新.

$1M = \frac{1,000,000}{250} \approx$ 4000个交易 → 10分钟. 每秒约7笔交易.



软分叉、Soft fork

假设更新 1M→0.5M



实际中可能出现软分叉的情况

修改 coinbase 中域的内容

背景：

① coinbase域中已经拿出8 bytes 去做了 extra nonce

② 轻节点如果要验证某交易是否在区块里，只需要全节点做一次 merkle proof 就可以

③ 全节点通过本地维护的 UTXO，可知道某账户的余额。

④ 轻节点无法知道某账户的余额

⑤ 有人建议，在将 utxo 取哈希，加入到 coinbase 的域中。

比特币历史上著名软分叉 case：P2SH Pay to Script Hash.



区块链技术与应用

第11讲：课堂问答

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

Q1：转账交易的时候如果接收者不在线怎么办

A：转账交易不需要接收者在线

Q2：私钥丢了怎么办？

A：丢了就没了。 Mt. Gox事件

有交易所的话，会由交易所保管，类似于现在中心化的机构，但会有风险

Q3：私钥泄露了怎么办

A：将账户所有的钱转到另一个账户

Q4：转账时写错地址怎么办

A：没有办法取消交易。转到不存在的地址（→ proof of burn），也没有办法不会退回。

Q5：既然所有写入区块的交易要被验证正确性，那为什么 proof of burn 中 OP_RETURN（无条件返回错误）会被区块接受。

A：对于某个交易，我们需要验证当前交易的输入脚本和之前交易的输出脚本。
而 OP_RETURN 是写在当前交易的输出脚本的，因此在本次验证不会被检查到

Q6：如何防止别的矿工偷答案。

A：在 coinbase tx 中，是有收款人地址的，要想偷答案就要修改地址，但修改地址会导致 Merkle Tree 发生改变，从而 Root hash 改变，从而 block header 发生改变，nonce 也不一样了。(作废了)

Q7：交易费怎么知道该给哪个矿工。

A：事先不需要知道哪个矿工。



区块链技术与应用

第12讲：比特币的匿名性

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

比特币匿名性没有我们想象的那么好。

Bit coin and anonymity

pseudonymity.

可能破坏比特币匿名性的几个方法

(一) 即使一笔交易生成多个 inputs 和 outputs. 这些 inputs 和 outputs 的地址也可能被人为关联.

	4BTC	5BTC
Inputs:	addr 1.	addr 2.
Outputs:	addr 3.	addr 4.
	6BTC	3BTC

① addr1 & addr2 一般会认为是一个人的.

② 可分析出 addr4 是找零钱地址, 那么和 addr1 & addr2 一般也是同一个人的.

(二) 地址账户和现实世界中的真实身份也可能产生关联

任何比特币世界与现实世界发生联系的时候, 都有可能泄露身份.

防范比特币洗钱: 盯住资金的转入转出链

eg Silk Road / Silk Road 2 ...

如何尽量提高匿名性

application layer → coin mixing: 把各种人混在一起.

— — —— — - -

network layer → 从节点的 ip 地址可以推算出真实身份

解决方案: 多路径转发

零知识证明

零知识证明是指一方（证明者）向另一方（验证者）证明一个陈述是正确的，而无需透露除该陈述是正确的外的任何信息。

e.g. 证明某个比特币账户是我的（不能泄露私钥）

用私钥产生一个签名来证明私钥是我的（公钥可以分发）

有争议到底是不是零知识证明

同态隐藏 零知识证明的数学基础

- 如果x,y不同，那么它们的加密函数值$E(x)$和$E(y)$也不相同。 → 如果$E(x)=E(y)$ 则$x=y$.
- 给定$E(x)$的值，很难反推出x的值。
- 给定$E(x)$和$E(y)$的值，我们可以很容易地计算出某些关于x,y的加密函数值。（同态运算）
  - 同态加法：通过$E(x)$和$E(y)$计算出$E(x+y)$的值
  - 同态乘法：通过$E(x)$和$E(y)$计算出$E(xy)$的值
  - 扩展到多项式

例子

证明者　验证者

- Alice想要向Bob证明她知道一组数x和y使得$x+y=7$，同时不让Bob知道x和y的具体数值。

简单的版本

- Alice把$E(x)$和$E(y)$的数值发给Bob
- Bob通过收到的$E(x)$和$E(y)$计算出$E(x+y)$的值（利用了性质3）
- Bob同时计算$E(7)$的值，如果$E(x+y)=E(7)$，那么验证通过，否则验证失败。

盲签方法

- 用户A提供SerialNum，（暗文）银行在不知道SerialNum的情况下返回签名Token，减少A的存款
- 用户A把SerialNum和Token交给B完成交易（明文）
- 用户B拿SerialNum和Token给银行验证，银行验证通过，增加B的存款（明文）
  → 银行不知道币的来源.
- 银行无法把A和B联系起来。
- 中心化

既让央行做中心化的Double Spending，又不让它知道交易的基本信息

BTC vs. Zerocoin

BTC：每一笔转账交易都要说明币的来源

Zerocoin：可以证明花的币是合法存在的，但不知道具体是哪一个

零币和零钞

- 零币和零钞在协议层就融合了匿名化处理，其匿名属性来自密码学保证。
- 零币(zerocoin)系统中存在基础币和零币，通过基础币和零币的来回转换，消除旧地址和新地址的关联性，其原理类似于混币服务。
- 零钞(zerocash)系统使用zk-SNARKs协议，不依赖一种基础币，区块链中只记录交易的存在性和矿工用来验证系统正常运行所需要关键属性的证明。区块链上既不显示交易地址也不显示交易金额，所有交易通过零知识验证的方式进行。



区块链技术与应用

第13讲：比特币引发的思考

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

一、Hash Pointer 指针只是形象的说法，并不是指针.

二、区块恋

共享，不要用截断私钥的方法，而要用多重签名的方法

↳ 会大大降低安全性

✔

三、分布式共识

为什么比特币系统能够绕过分布式共识中的那些不可能结论?

比特币并没有取得严格意义的共识：如分叉.

✔

四、比特币稀缺性.

五、量子计算.

量子计算的超大算力是否会对比特币等加密货币的密码学基础产生影响.?

1). 量子计算离应用还很远（有生之年...）

2). 量子计算首先冲击传统金融业.

3) 加密和取哈希是2个不同的操作，即使量子计算机也无法完成逆取哈希.



区块链技术与应用

第14讲：以太坊概述

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

相对于比特币的几点改进

① 缩短出块时间至10多秒.

② ghost 共识机制

③ mining puzzle.
- BTC：计算密集型
- ETH：memory-hard.（限制ASIC）

④.（未来）用pos代替pow.

⑤ 对智能合约的支持.

BTC： decentralized currency

ETH： decentralized contract. → 优势、多主体，司法管辖权受限 e.g. 众筹.



区块链技术与应用

第15讲：以太坊的账户

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

account-based ledger

BTC账户：tx-based ledger → 要想知道余额，要去UTXO找关联账户并加总

验证

A→B (10 BTC)　　　B→C (3BTC)
　　　　　　　　　└→B′ (7BTC)

BTC的转账，具体说明了哪几个币转。

ETH账户：account-based ledger

A→B (10 BTC)　　　B→C (3BTC)

ETH的转账，不需要具体说明哪几个币转。

account-based 模式的优点

① 符合人主观感受，和现行银行交易方式类似

② 可有效防范 Double Spending Attack.

account-based 模式的缺点

① replay attack

A→B (10 ETH)　A给B转账10个ETH，B有恶意，他将这笔交易重放了一次
A→B (10 ETH).　其它节点会认为是一笔新的交易.

解决方案：加 nonce (交易次数编号).

A→B (10 ETH)
nonce = 21
signed by A

→ 因为有A的签名，该交易只能重放，但有nonce保护，由此防止发生 replay attack

Externally owned account

所有的交易，只能由外部账户发起，不能由合约账户发起

* balance
* nonce（计数器，不是随机的）

Smart contract account 合约账户

* balance
* nonce（一个合约可以调用另一个合约，因此也要通过nonce记录调用的次数）
* code
* storage

为什么要设置ETH这个全新的体系

智能合约要求有稳定的账户参与。



区块链技术与应用

第16讲：以太坊中的状态树

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

目标：一个映射：从账户地址 ⟶ 账户状态

addr (账户地址) ⟶ state (账户状态)
160 bits
40个16进制数.

思路1：使用(key, value)的 hash table 储存 addr 和 state
将 hash table 组成 merkle Tree，Tree 的 root hash
保存在 Block header 中.

注意：比特币的mt记录的是tx.
ETH 一mt 一记录的是 account.

存在问题：每次出块会有新交易打包进块中，从而改变 merkle tree.
但事实上只有一部分账户发生改变
一小部分改变要重写整个tree，有点得不偿失.

另外，由于BTC打包tx，数量级没那么大(一段时间内交易数量有限)
而ETH打包账户，数量级就指数级上升(每次都必须打包所有账户)

思路2：直接用 merkle tree 存放账户，要改内容直接改 merkle tree，是否可行

存在问题：merkle 没有提供一个高效的查找和更新的方法

比特币运行方式：

每个节点在本地运行一个候选区块，(每个节点记录的交易，顺序都不同)
每个节点去挖矿，去竞争记账权。取得记账权的节点发布的区块中的交易为该区块链记录的交易.

因此，顺序唯一，不需要排序.

为什么ETH不能这么做？因为如果由发布区块的节点发布账户信息，发布的仍然是所有账户的信息(思路1问题)

思路3：使用sorted Merkle Tree 是否可行？

回答：新增账户，产生的账户地址是随机的，就要插入并重新排序

MPT结构

引例：trie结构（字典树）

trie特点：

①打乱顺序，trie结构不变。

天然排序，即使插入新值，而不影响。
适用于ETH的account-base.

②具有很好的更新局部性。

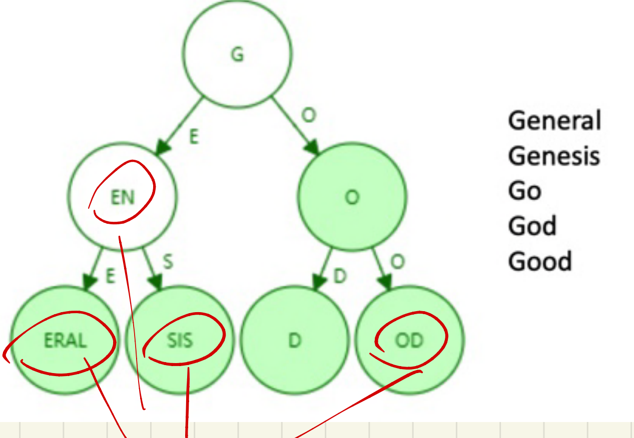
如右图，更新general不需要遍历整棵树

General
Genesis
Go
God
Good

trie缺点：存储浪费，部分内容效率低（键值分布稀疏）

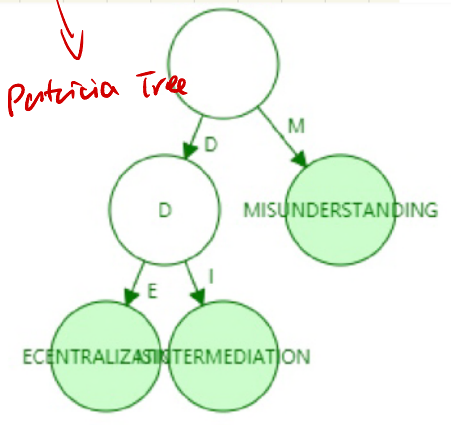
解决方案：

Patricia Tree（经过路径压缩的树）



misunderstanding
decentralization
disintermediation

Patricia Tree



压缩 ⟹ 树的高度明显减少。

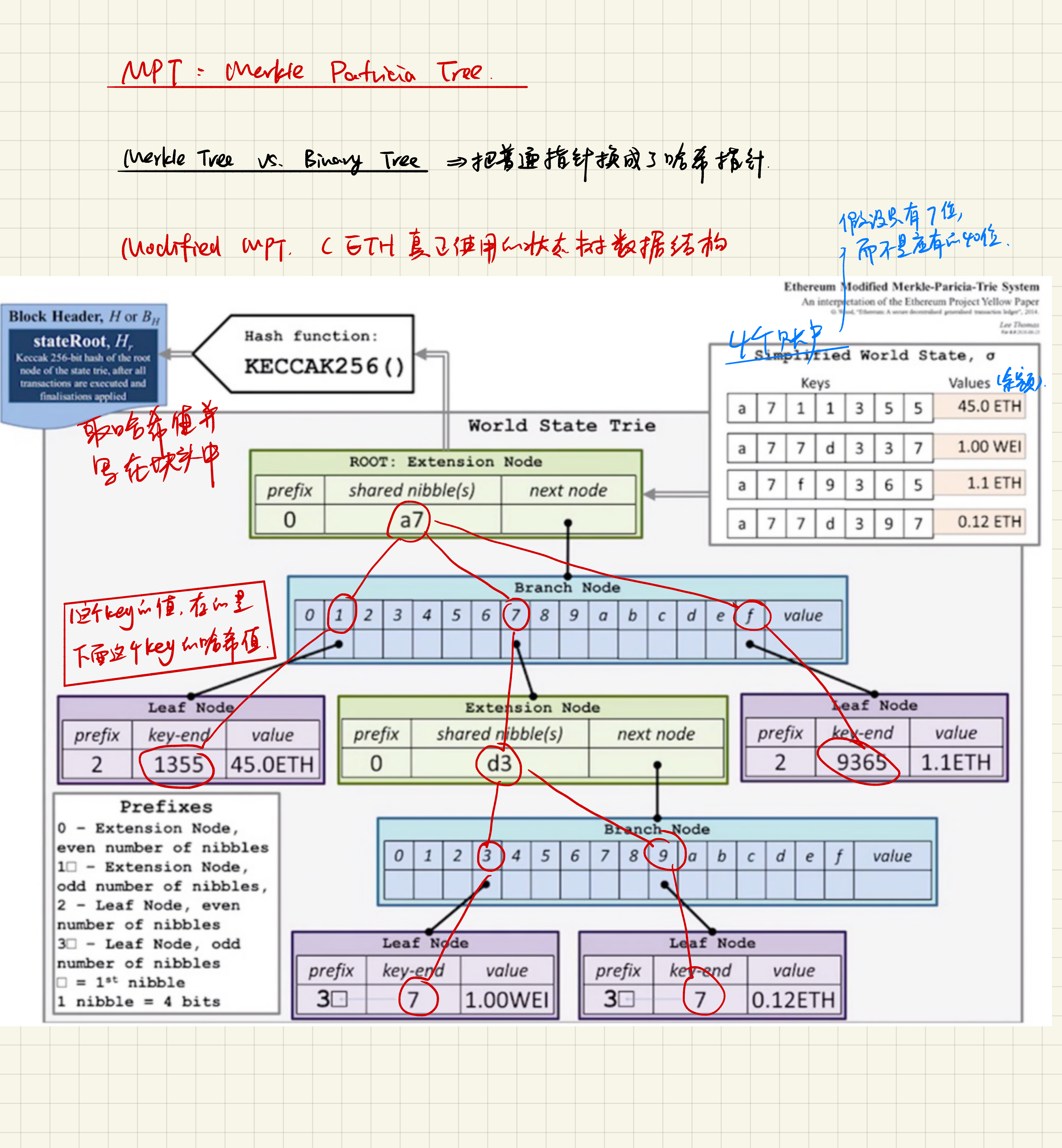
ETH的地址是40位十六进制数，共160 bits. 因此地址共有 2^{160} 种. 从而其键值分布是极其稀疏的.

（地址设置成 2^{160} 这么大，目的是防止 hash collision）

MPT: Merkle Patricia Tree.

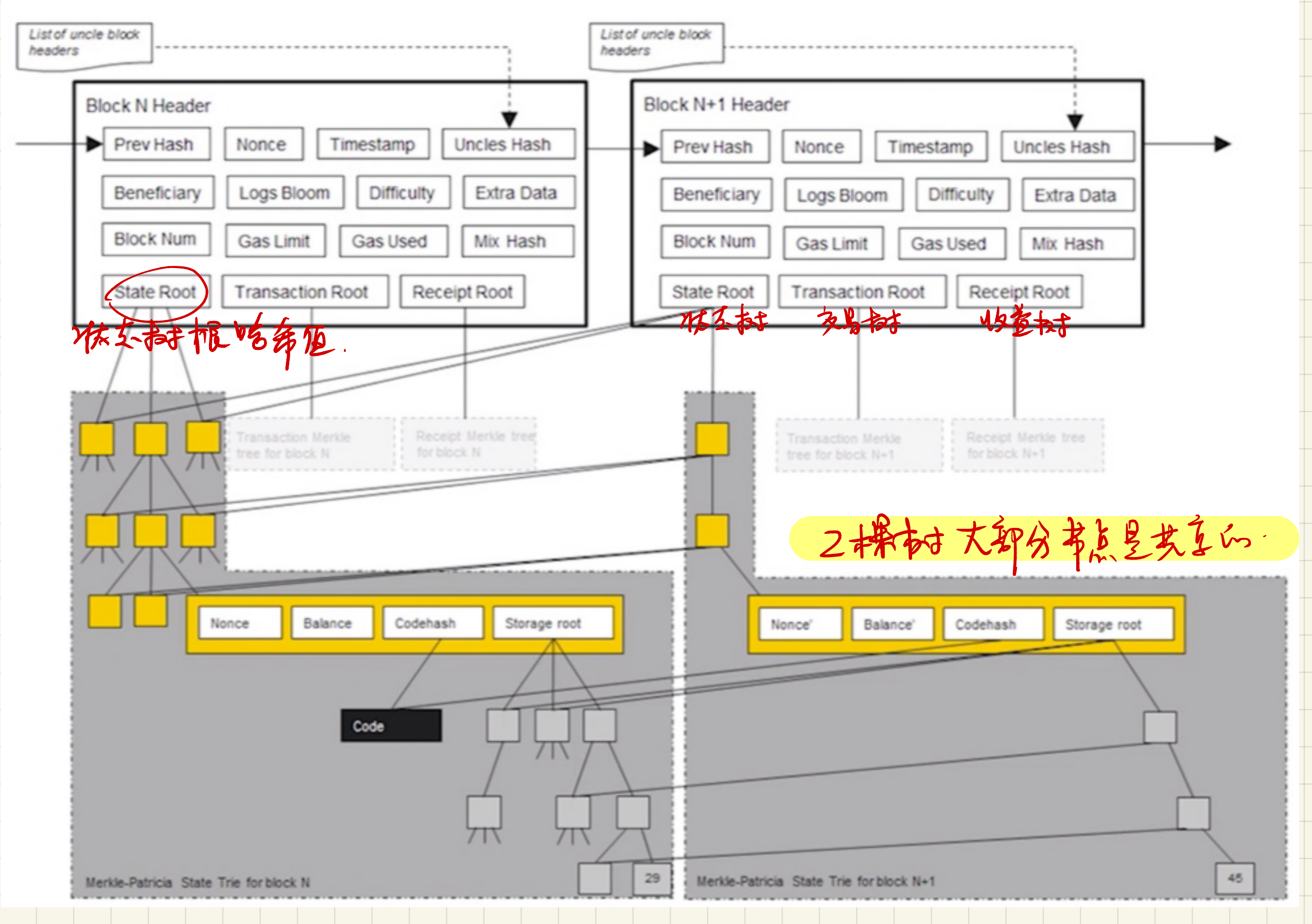
Merkle Tree vs. Binary Tree ⇒ 把普通指针换成了哈希指针.

Modified MPT. (ETH 真正使用的状态树数据结构



每次发布新的区块时，状态树中新节点的值会发生变化，这些改变不是在原地改，而是新建一些分支，原来的状态其实是保留下来的。

下图：2个相邻的区块



问题：为什么要保留历史状态，不在原先数据上直接修改。

答：为了回滚。在ETH，分叉是常态，orphan block中的数据都要向前回滚。而由于ETH中有智能合约，为了支持智能合约的回滚，必须保持之前状态

```go
69  // Header represents a block header in the Ethereum blockchain.
70  type Header struct {
71  	ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
72  	UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
73  	Coinbase    common.Address `json:"miner"            gencodec:"required"`
74  	Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
75  	TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
76  	ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
77  	Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
78  	Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
79  	Number      *big.Int       `json:"number"           gencodec:"required"`
80  	GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
81  	GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
82  	Time        *big.Int       `json:"timestamp"        gencodec:"required"`
83  	Extra       []byte         `json:"extraData"        gencodec:"required"`
84  	MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
85  	Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
86  }
```

> Coinbase: 挖出区块的矿工的地址.
> Root / TxHash / ReceiptHash: 三棵树的根hash.
> Difficulty: 难度.
> GasLimit / GasUsed: 智能合约消耗的汽油费

状态树中保存的是 (key, value). 上面讲的都是 key 保存的地址.
那么value呢?

经过 RLP 序列化, 再存储

↳ Recursive Length Prefix.



区块链技术与应用

第17讲：以太坊中的交易树和收据树

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

都是MPT.

→ 每次发布新区块时，区块中的交易组织成交易树

→ 每个交易执行完会形成一个收据，记录该交易的相关信息

交易树和收据树的用处

① 提供 merkle proof.

② 支持更复杂的查询操作. → e.g. 支持查找过去10天和某个智能合约相关的交易

↳ 解决方案：引入了 bloom filter 数据结构 (有可能出现false positive，而不会出现 false negative).

布隆过滤器的原理是，当一个元素被加入集合时，通过K个散列函数将这个元素映射成一个位数组中的K个点，把它们置为1。检索时，我们只要看看这些点是不是都是1就（大约）知道集合中有没有它了：如果这些点有任何一个0，则被检元素一定不在；如果都是1，则被检元素很可能在。这就是布隆过滤器的基本思想。

S { a b c } n=3

```
for x in S
    for i in {1..3}
        B[hi(x)]=1;
```

h1(a) h2(a) h3(a) h1(b) h2(b) h3(b) h1(c) h2(c) h3(c)

B | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

m=28

h1(d) h2(d) h3(d) h1(e) h2(e) h3(e)

```
for i in {1..3}
    if B[hi(x)]==0
        return false;
return true;
```

d 可能在集合S中，也可能不在
(出现哈希碰撞).

e 一定不在集合S中
因为有1个0，所以一定不在.

bloom filter 另一个局限性：不支持删除操作（还是哈希碰撞的原因）.

ETH 有关 bloom filter 的具体工作

每个交易执行完后，会形成一个收据，收据中包含一个 bloom filter，记录交易的类型、地址等其它信息

发布的区块在块头中，也有一个总的 bloom filter，这个总的 bloom filter，是该区块中所有交易的 bloom filter 的并集

当我们要查找过去 10 天发生的某智能合约相关的所有交易，方法是：

Step 1：找哪个区块的块头中的 bloom filter 有我们需要的交易的类型。如果块头中没有，说明这个区块不是我们想要的（并集）

Step 2：如果块头有，就去查这个区块各收据的 bloom filter

ETH 的运行过程：交易驱动的状态机（tx-driven state machine）

tx → 每次发布的区块中包含的 tx

state → 状态树中账户的状态

通过执行交易，系统从当前状态转移到下一个状态

UTXO 是 BTC 的 state machine

问题：某人在 ETH 发布一个交易，某节点收到这个 tx。tx：A → B，有无可能 B 的地址以前从来没有过？

答：有可能，创建账户不需广播，只有到收到转账才有可能。

问题：现行状态树的机制是，包含全部的账户的状态，可否改成，仅包括区块中涉及的 tx 的账户的状态

答：① 查找某账户的状态不方便，得往前面的区块找。如 A → B 的交易，要验证 A 的余额，如果 A 很长一段时间内没有交易，就需要往前找很长

② 更严重的问题：A → B（10 ETH），还要找 B 的账户状态（因为要往上加 10 个 ETH）。如果 B 是新建账户，就会一直找到 genisis block，发现 genisis block 中都没有 B，才能确认 B 是新建账户

代码实现见 Notability.



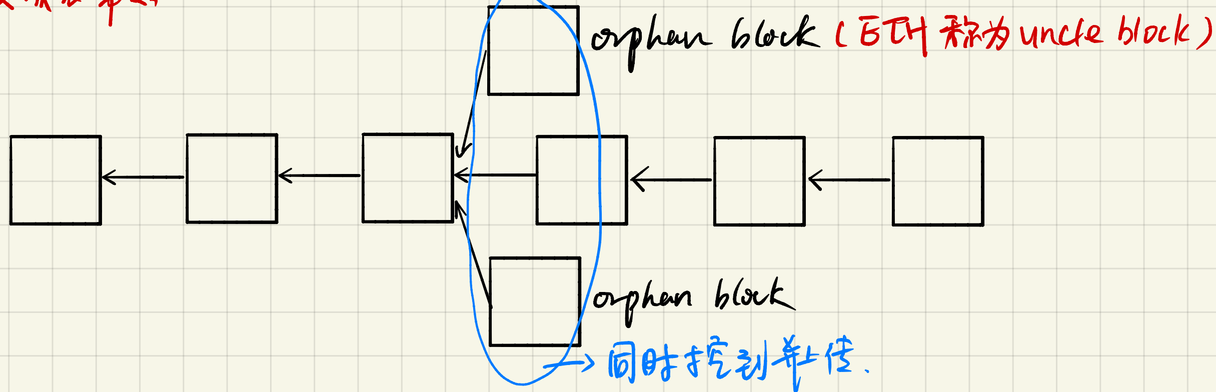
区块链技术与应用

第18讲：GHOST协议

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

ETH将区块生成时间设定为10几秒可能带来以下的问题

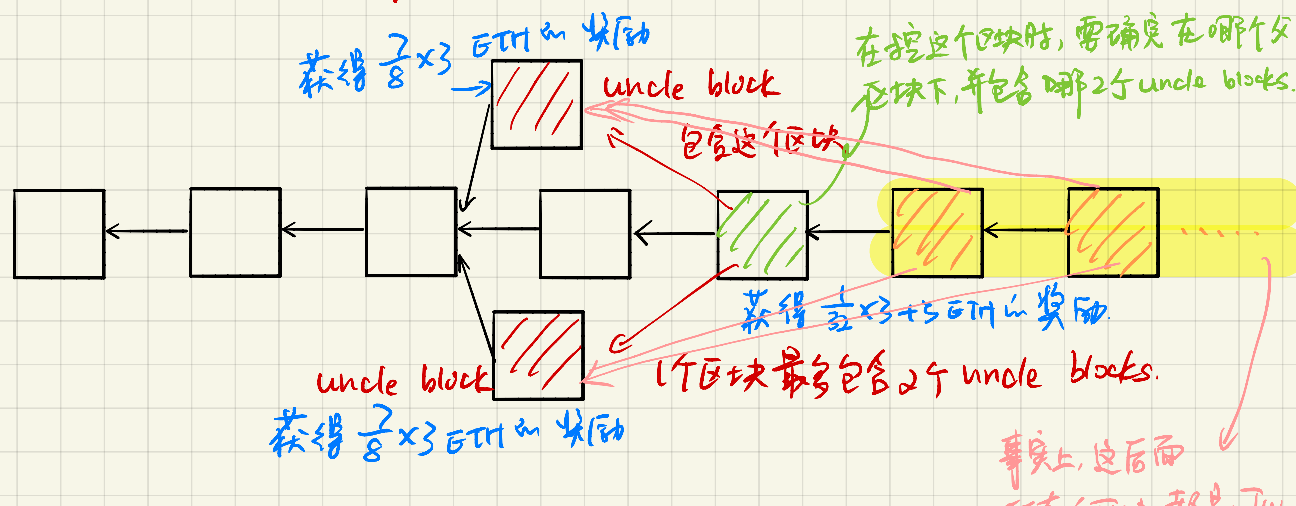
① 出块时间10几秒，同时由于 Network layer 限制，一个区块发布到其它节点，也需要10几秒的时间，从而分叉成为常态。



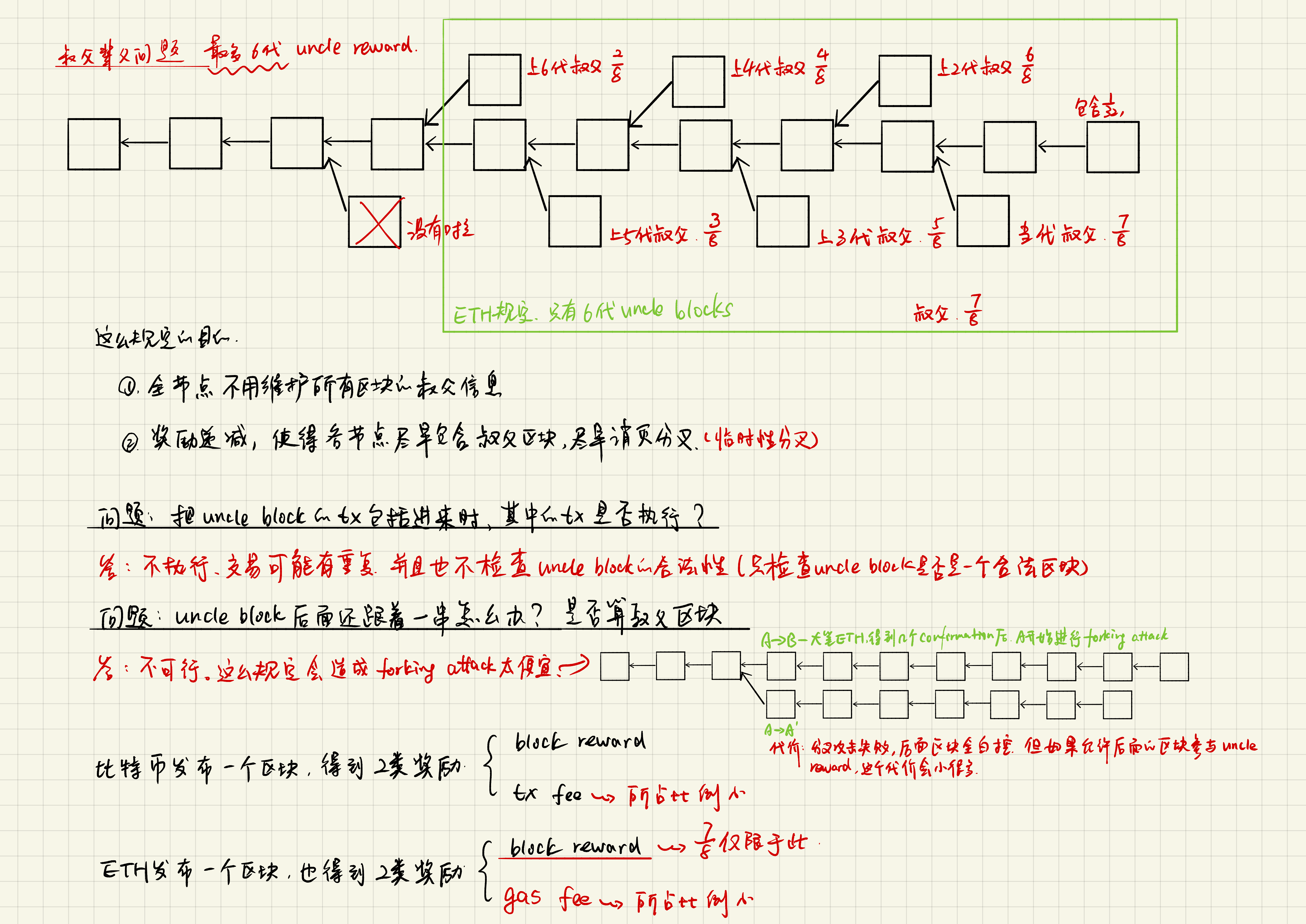
对比特币来说，由于出块时间较长，上述最长合法链的机制尚可接受。

但对ETH来说，出块时间太短，分叉常态了，上述机制就不再适用了。
对个体矿工尤其不公平，因为大矿场有能力去制造最长合法链 ↑ mining centralization /centralization bias.

GHOST协议核心：给挖到orphan blocks的节点"安慰奖"



GHOST协议允许uncle blocks "记辈分"的原因在于鼓励子区块包含父区块


叔父辈问题 最多6代 uncle reward.
上6代叔父 2/8
上4代叔父 4/8
上2代叔父 6/8
包含者,
没有啦
上5代叔父 3/8
上3代叔父 5/8
当代叔父 7/8
ETH规定, 只有6代 uncle blocks
叔父 7/8
这么规定的目的.
①. 全节点不用维护所有区块的叔父信息
②. 奖励递减, 使得各节点尽早包含叔父区块, 尽早消灭分叉. (临时性分叉)
问题: 把 uncle block 的 tx 包括进来时, 其中的 tx 是否执行?
答: 不执行. 交易可能有重复. 并且也不检查 uncle block 的合法性 (只检查 uncle block 是否是一个合法区块)
问题: uncle block 后面还跟着一串怎么办? 是否算叔父区块
答: 不可行. 这么规定会造成 forking attack 太便宜.
A->B 一大笔ETH, 得到n个 confirmation 后. A开始进行 forking attack
A->A'
代价: 分叉攻击失败, 后面区块全白挖. 但如果允许后面的区块参与 uncle reward, 这个代价会小很多.
比特币发布一个区块, 得到2类奖励
block reward
tx fee 所占比例小
ETH发布一个区块, 也得到2类奖励
block reward 7/8 仅限于此.
gas fee 所占比例小

ETH真实情况

Etherscan The Ethereum Block Explorer —— 这个网站有ETH的状态情况

LOGIN | Search by Address / Txhash / Block / Token / Ens | GO

HOME | BLOCKCHAIN | TOKENS | RESOURCES | MISC



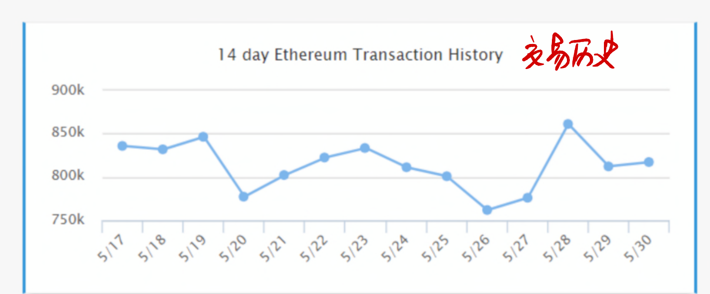
MARKET CAP OF $56.950 BILLION
$570.75 @ 0.07597 BTC/ETH (+0.58%)

LAST BLOCK
5708456 (14.7s)

TRANSACTIONS
239.32 M (10.7 TPS)

Hash Rate
275,805.81 GH/s

Network Difficulty
3,209.03 TH



Blocks 最新区块 | View All

Block 5708456
> 39 secs ago
Mined By DwarfPool1
211 txns in 37 secs
Block Reward 3.3066 Ether

Block 5708455
> 1 min ago
Mined By Ethermine
179 txns in 26 secs
Block Reward 3.40638 Ether

Block 5708454
Mined By Ethermine

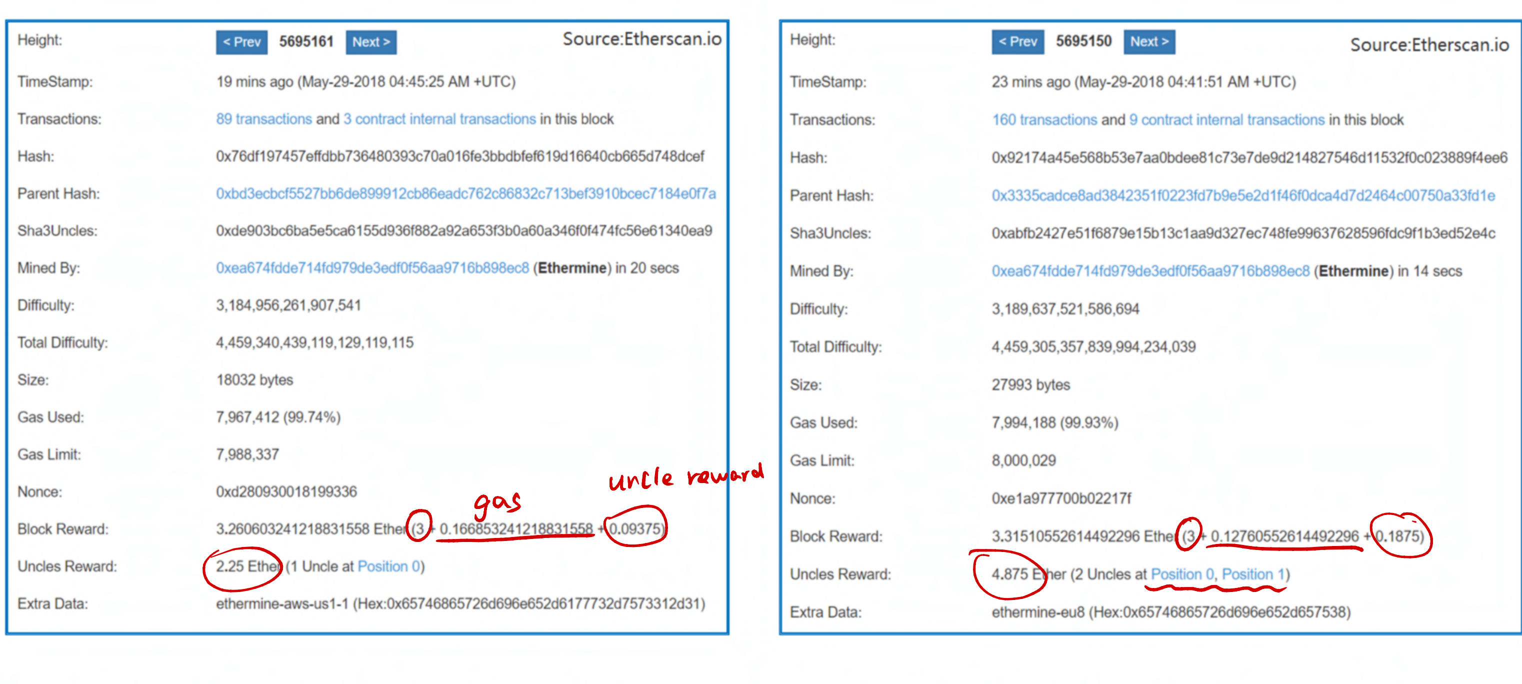


叔父区块的几种情况

Block Height	UncleNumber	Age	Miner	Reward	
5695161 —	5695159 =2 说明是上2代叔父	4 mins ago	f2pool_2	2.25 Ether	#1
5695159	5695157	5 mins ago	miningpoolhub_1	2.25 Ether	
5695157	5695155	6 mins ago	0xb6b12f9f4ed7c57…	2.25 Ether	
5695154 —	5695153 =1 说明是当代叔父	7 mins ago	Ethermine	2.625 Ether	#2
5695150	5695148	8 mins ago	bitclubpool	2.25 Ether	
5695150	5695149	8 mins ago	f2pool_2	2.625 Ether	
5695142	5695141	9 mins ago	Nanopool	2.625 Ether	
5695133	5695131	11 mins ago	f2pool_2	2.25 Ether	
5695129	5695128	12 mins ago	Nanopool	2.625 Ether	
5695119	5695118	15 mins ago	0x92e3f585ab69944…	2.625 Ether	
5695113	5695111	16 mins ago	Nanopool	2.25 Ether	
5695109 —	5695106 =3 说明是上3代叔父	17 mins ago	DwarfPool1	1.875 Ether	#3

Source:Etherscan.io

距离	比例	实际奖励	示例
1	7/8	2.625	#2
2	6/8	2.25	#1
3	5/8	1.875	#3
4	4/8	1.5	
5	3/8	1.125	
6	2/8	0.75	



引入1个uncle的reward是$3 \times \frac{1}{32} = 0.09375$

引入2个uncle的reward是$2 \times 3 \times \frac{1}{32} = 0.1875$



北京大学 PEKING UNIVERSITY

区块链技术与应用

第19讲：以太坊的挖矿算法 → ethash

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

Block chain is secured by mining

BTC mining 饱受争议的地方在于：需用专业的ASIC芯片，这与去中心化理念背道而驰。

↓

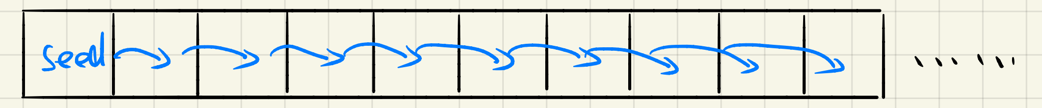
其它加密货币：ASIC resistance

↓

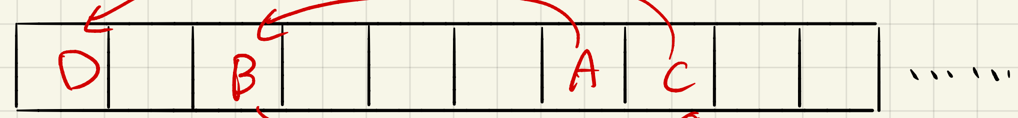
e.g. LiteCoin (基于 Scrypt) ← 方式：① memory hard mining puzzle.

Scrypt：一种对内存要求很高的哈希函数。

Step1：在数组中首位填入随机数，之后的数通过 hash function 算出来并依次……



Step2：开始按顺序读取数据解 puzzle



为了高效挖矿，需要保存内存区域，但同时也带来问题，即：轻节点也需要保存上述数组，很占内存。

ETH 的改进，2种数据集

- 16M cache → 轻节点，便于验证
- 1G dataset → 全节点

过程：

16M cache 生成方式与 Scrypt 类似

Step1：在数组中首位填入随机数，之后的数通过 hash function 算出来并依次……

与LiteCoin类似

seed ……

Step2: 生成一个更大的数组 (远大于之前那个) → 即 1G 的 dataset.

从前后cache中按照一定顺序、算法读出256个数，再组成一个数放在这个位置。

执行类似操作

执行类似操作，直到填满这个数组。

Step3: 找nonce，通过nonce找位置(及其相邻)，再经过运算找位置(及其相邻)，往复64次，共找128个数，取哈希，与block header中的target比较，直到满足target要求

③ … 64次，128个数. ① ②

代码实现: (python).

Step1: 生成 16M cache

```python
def mkcache(cache_size, seed):
    o = [hash(seed)]
    for i in range(1, cache_size):
        o.append(hash(o[-1]))
    return o
```

→ 创建o这个集合.

→ for 循环，往里加元素.
hash(o[-1]) ~ 前一个值的哈希值.

每隔30000个块会重新生成seed（对原来的seed求哈希值），并且利用新的seed生成新的cache。

cache的初始大小为16M，每隔30000个块重新生成时增大初始大小的1/128 —— 128K。

Step2: 通过cache，生成dataset中的第i个元素

```python
def calc_dataset_item(cache, i):
    cache_size = cache.size
    mix = hash(cache[i % cache_size] ^ i)
    for j in range(256):
        cache_index = get_int_from_item(mix)
        mix = make_item(mix, cache[cache_index % cache_size])
    return hash(mix)
```

get_int_from_item: 用当前算出来的哈希值，读出下一个位置

make_item: 用cache中第i个元素的值，和当前的哈希值，计算出下一个哈希值.

Step3: 多次调用这个函数，就可以得到完整的dataset。

矿工挖矿的算法实现

header：块头
nonce：当前尝试的随机值、
full_size：当前dataset中元素的个数
dataset：当前dataset、

```
def hashimoto_full(header, nonce, full_size, dataset):
    mix = hash(header, nonce)
    for i in range(64):
        dataset_index = get_int_from_item(mix) % full_size
        mix = make_item(mix, dataset[dataset_index])
        mix = make_item(mix, dataset[dataset_index + 1])
    return hash(mix)
```

先通过header和nonce求出一个初始的mix，然后进入64次循环，根据当前的mix值求出要访问的dataset的元素的下标，然后根据这个下标访问dataset中两个连续的的值。→ 结合上一页逻辑看。

矿工挖矿主循环

```
def mine(full_size, dataset, header, target):
    nonce = random.randint(0, 2**64)
    while hashimoto_full(header, nonce, full_size, dataset) > target:
        nonce = (nonce + 1) % 2**64
    return nonce
```

轻节点验证算法实现

轻节点任务：验证这个nonce是否符合要求

```
def hashimoto_light(header, nonce, full_size, cache):
    mix = hash(header, nonce)
    for i in range(64):
        dataset_index = get_int_from_item(mix) % full_size
        mix = make_item(mix, calc_dataset_item(cache, dataset_index))
        mix = make_item(mix, calc_dataset_item(cache, dataset_index + 1))
    return hash(mix)
```

问题：为什么矿工要保存所有的dataset，而轻节点只需保存cache？

由于矿工需要验证非常多的nonce，如果每次都要从16M的cache中重新生成的话，那挖矿的效率就太低了，而且这里面有大量的重复计算：随机选取的dataset的元素中有很多是重复的，可能是之前尝试别的nonce时用过的。所以，矿工采取以空间换时间的策略，把整个dataset保存下来。轻节点由于只验证一个nonce，验证的时候就直接生成要用到的dataset中的元素就行了。

ETHER DISTRIBUTION OVERVIEW

Genesis (60M Crowdsale+12M Other):	72,009,990.50 Ether
+ Mining Block Rewards:	25,903,969.97 Ether
+ Mining Uncle Rewards:	1,818,301.25 Ether
= Current Total Supply	**99,732,261.72 Ether**

总供应量约1亿枚

Data Source: Total Eth Supply API

$ PRICE PER ETHER

In USD:	$518.77
In BTC:	0.07321

市价

Data Source: CryptoCompare

99,732,261.72
Total Ether Supply

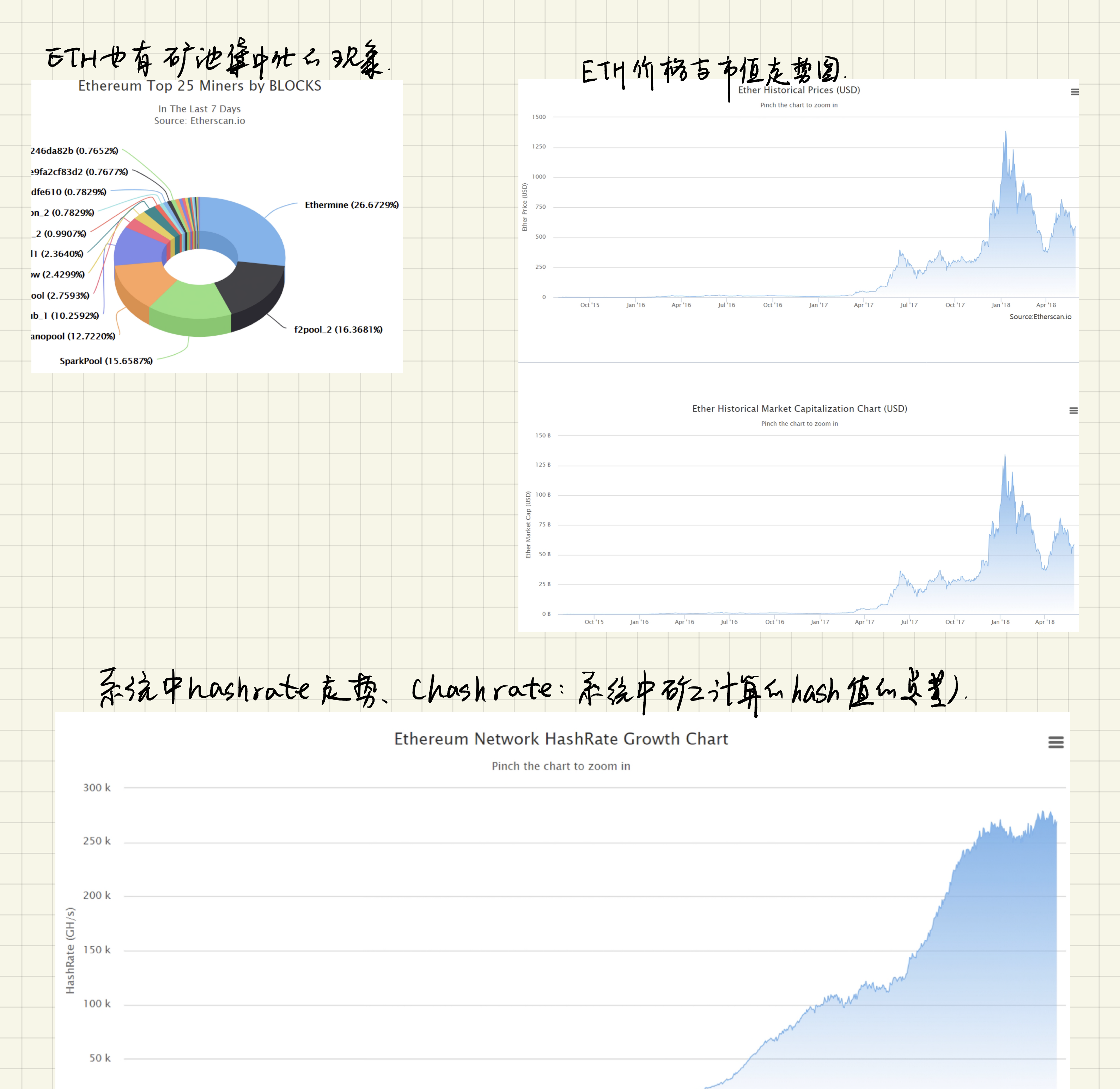
$51,738,105,411
Market Capitalization
总市值

Breakdown By Supply Types

市值分布

Genesis (72009990.49948 ETH)
Block Rewards (25903969.9688 ETH)
Uncle Rewards (1818301.25 ETH)

Source:Etherscan.io



最后一问：有人认为，ASIC一统天下比用通用计算机挖矿要好，原因是什么？

答：如果挖矿必须使用ASIC，则发动攻击就必须购买大量ASIC。ASIC是专用设备，除了挖某种货币以外不能做其它事。因此，发动攻击的成本极高（因为一旦攻击成功，该货币价值会大幅下降，相当于损人不利己）。

但如果可以用通用设备挖矿，那么发动攻击的成本将大幅下降。



区块链技术与应用

第20讲：以太坊的挖矿难度调整

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

比特币难度：每隔2016个区块调整一次难度，使得出块时间维持在10分钟左右.
ETH：每个区块都有可能调整挖矿难度，调整方法比较复杂，而且改过多个版本

ETH现行算法

PART 1：主公式

$$D(H) \equiv \begin{cases} D_0 & \text{if} \quad H_i = 0 \\ \max(D_0, P(H)_{H_d} + x \times \varsigma_2) + \epsilon & \text{otherwise} \end{cases}$$

where:

$$D_0 \equiv 131072 \quad (42)$$

区块当前难度（$D(H)$）；H_i：区块的序号；$\max(D_0, P(H)_{H_d} + x \times \varsigma_2)$：基础部分；$\epsilon$：难度炸弹，为了向PoS过渡；$D_0$：难度下限

- $P(H)_{Hd}$为父区块的难度，每个区块的难度都是在父区块难度的基础上进行调整。
- $x \times \varsigma_2$ 用于自适应调节出块难度，维持稳定的出块速度。
- ϵ 表示设定的难度炸弹。

PART2：自适应难度调整 $x \times \varsigma_2$

$$x \equiv \left\lfloor \frac{P(H)_{H_d}}{2048} \right\rfloor \quad (43)$$

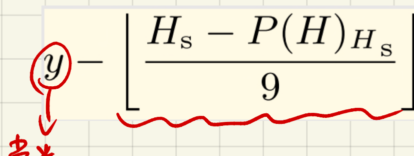
$$\varsigma_2 \equiv \max\left(y - \left\lfloor \frac{H_s - P(H)_{H_s}}{9} \right\rfloor, -99\right) \quad (44)$$

-99：难度调整上界

y和父区块的uncle数有关。如果父区块中包括了uncle，则y为2，否则为1。

- 父块包含uncle时难度会大一个单位，因为包含uncle时新发行的货币量大，需要适当提高难度以保持货币发行量稳定。

难度调整逻辑.



y：常数

如果上式为正，则增加难度，反之减小难度.

H_s是本区块的时间戳，$P(H)_{H_s}$是父区块的时间戳，均以秒为单位，并规定$H_s > P(H)_{H_s}$。$H_s - P(H)_{H_s}$为出块间隔

- 该部分是稳定出块速度的最重要部分：出块时间过短则调大难度，出块时间过长则调小难度。

难度调整举例

以父块不带uncle的情况$(y = 1)$为例：

- 出块时间在[1,8]之间，出块时间过短，难度调大一个单位。
- 出块时间在[9,17]之间，出块时间可以接受，难度保持不变。
- 相差时间在[18,26]之间，出块时间过长，难度调小一个单位。
- ...

PART3: 难度炸弹 ε

$$\epsilon \equiv \left\lfloor 2^{\left\lfloor H'_i \div 100000 \right\rfloor - 2} \right\rfloor$$

$$H'_i \equiv \max(H_i - 3000000, 0)$$

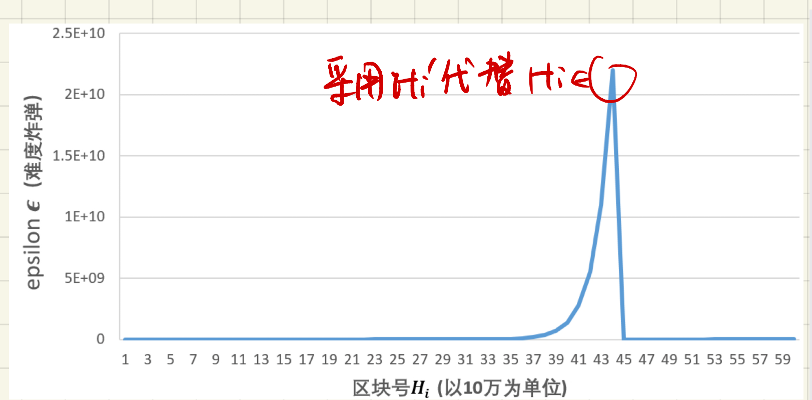
H_i → 当前区块序号

→ 指数级：随着出块量增加，H_i / H'_i 值越来越大，从而 ε 越来越大，且呈指数级增长

为什么设置难度炸弹？

- 设置难度炸弹的原因是要降低迁移到PoS协议时发生fork的风险：到时挖矿难度非常大，所以矿工有意愿迁移到PoS协议。

插曲：PoS共识达成协议难度大于预期，而 ε 已经增大导致挖矿时间大幅上升，于是ETH决定用 H'_i 代替 H_i，倒退300万个区块，为PoS达成协议赢得时间。



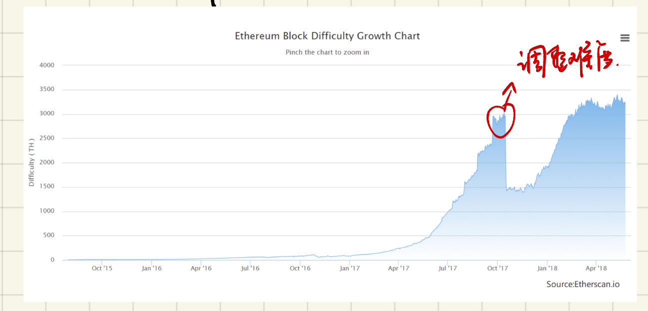
以太坊发展的四个阶段

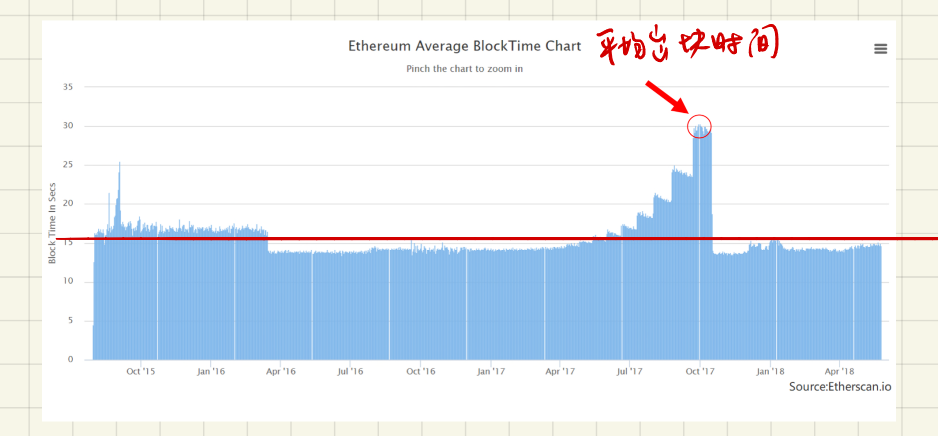
- Frontier
- Homestead
- Metropolis
  - Byzantium（当前阶段）
  - Constantinople
- Serenity

➢ 说明

- Metropolis又分为Byzantium和Constantinople两个子阶段。
- 难度炸弹的回调发生在Byzantium这个子阶段，在EIP（Ethereum Improvement Proposal）中决定，同时把block reward从5个ETH降为3个ETH。

ETH的统计数据







北京大学 PEKING UNIVERSITY

区块链技术与应用

第21讲：权益证明　PoS = proof of stake.

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

PoW饱受批评的一点在于浪费资源



Key Network Statistics

Description	Value
Bitcoin's current estimated annual electricity consumption* (TWh)	69.95
Annualized global mining revenues	$6,084,977,937
Annualized estimated global mining costs	$3,497,452,665
Current cost percentage	57.48%
Country closest to Bitcoin in terms of electricity consumption 相当于智利全年用电量	Chile
Estimated electricity used over the previous day (KWh)	191,641,242
Implied Watts per GH/s	0.207
Total Network Hashrate in PH/s (1,000,000 GH/s)	38,662.00
Electricity consumed per transaction (KWh) 一个交易的能耗	1,014
Number of U.S. households that could be powered by Bitcoin 647万户 US family	6,476,764
Number of U.S. households powered for 1 day by the electricity consumed for a single transaction	34.26
Bitcoin's electricity consumption as a percentage of the world's electricity consumption	0.31%
Annual carbon footprint (kt of CO2) 占全球用电量0.31%	34,275
Carbon footprint per transaction (kg of CO2)	496.79

问题：挖矿消耗的能耗是否是必须的

挖矿的本质：大家投钱买矿机，比拼算力，算力越大，挖矿概率越高

因此，本质为：投入的钱越多，挖矿概率越高

PoS核思想：（不再用钱买矿机拼算力，而是直接拼钱 ⟶ virtual mining

PoS vs. PoW 优点：

① 节能。

☆ ② PoS是闭环生态，而PoW是开放生态，因此PoS天然防止了51% attack.

理解：PoW下，attacker可以在现实世界中购买矿机，来增加算力以达到51% attack目的。而PoS下，attacker必须购买足够多的加密货币（相当于成为股东），才有发动attack的能力，但此时，对于币的开发者和早期矿工其实是受益的。（如同恶意收购）。

PoS 和 PoW 并不互斥。

nothing at stake：当出现下述分叉时 PoW下会选择一条链挖，而PoS下2条链都可以下注。

ETH准备采用的PoS协议为 Casper the Friendly Finality Gadget (FFG)

引入了：validator → 投入一定数量保证金。

每挖出100个区块生成一个epoch，然后开始投票 (two-phase commit)

100 → 现在减少为50.

2轮投票 { 2/3 prepare message ; 2/3 Commit Message.

原始casper规定

epoch

100个

结束时，2轮投票 { 2/3 prepare ; 2/3 Commit

现行casper规定

epoch　epoch

50个　50个

2/3　2/3

结束时1轮投票，这1轮投票，对于前一个epoch，是commit message，对于后一个epoch，是prepare message.

对于validator来说，积极参与投票有奖励，行政不作为则会受到处罚（扣保证金）

而对于2边下注的情况，要没收全部保证金



区块链技术与应用

第22讲：智能合约

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

什么是智能合约？

- 智能合约是运行在区块链上的一段代码，代码的逻辑定义了合约的内容
- 智能合约的帐户保存了合约当前的运行状态
  - balance：当前余额
  - nonce：交易次数
  - code：合约代码
  - storage：存储，数据结构是一棵MPT
- Solidity是智能合约最常用的语言，语法上与JavaScript很接近

代码示例：

```
pragma solidity ^0.4.21;                                    版本号

contract SimpleAuction {                                    定义状态变量
    address public beneficiary;      //拍卖受益人
    uint public auctionEnd;          //结束时间
    address public highestBidder;    //当前的最高出价人
    mapping(address => uint) bids;   //所有竞拍者的出价
    address[] bidders;               //所有竞拍者

    // 需要记录的事件
    event HighestBidIncreased(address bidder, uint amount);
    event Pay2Beneficiary(address winner, uint amount);

    /// 以受益者地址 `_beneficiary` 的名义，
    /// 创建一个简单的拍卖，拍卖时间为 `_biddingTime` 秒。
    constructor(uint _biddingTime,address _beneficiary
        ) public {
        beneficiary = _beneficiary;
        auctionEnd = now + _biddingTime;
    }

    /// 对拍卖进行出价，随交易一起发送的ether与之前已经发送的
    /// ether的和为本次出价。
    function bid() public payable {…          → 能接受外部转账的标记
    }

    /// 使用withdraw模式
    /// 由投标者自己取回出价，返回是否成功
    function withdraw() public returns (bool) {…
    }

    /// 结束拍卖，把最高的出价发送给受益人
    function pay2Beneficiary() public returns (bool) {…
    }
}
```

address、uint 都是数据类型

mapping：哈希表

solidity里数组是动态的，这里bidders是一个数组

bidders.push(bidder) … 增加元素

bidders.length 求数组长度

event 事件

用constructor构造函数

constructor只能有一个

3个成员函数

问题：外部账户如何调用智能合约.

何为调用？

对于 A->B 这个交易，如果B是一个账户，则这笔tx是一次普通转账
而如果B是一个智能合约账户，则这笔tx就是一次A对B合约的调用

创建一个交易，接收地址为要调用的那个智能合约的地址，data域填写要调用的函数及其参数的编码值。

← BACK   TX 0x73275297b391f3e08b1cc7144d7ab5fcf77fecee92b46ca9ec2946f56ebf8ea2

SENDER ADDRESS 发起人地址		TO CONTRACT ADDRESS 合约地址		CONTRACT CALL
0x903db0EbD4206669Ab50BCF93c550df9b5Da178c		0x5E31d519A6F34d224C25B706687EE2AbF170B888		
VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	21657	1000000000	6000000	3

TX DATA
0x2a24f46c



问题：一个合约如何调用另一个合约中的函数.

1. 直接调用（A、B为合约）

```
3  contract A {
4      event LogCallFoo(string str);
5      function foo(string str) returns (uint){
6          emit LogCallFoo(str);
7          return 123;
8      }
9  }
10
11 contract B {
12     uint ua;
13     function callAFooDirectly(address addr) public{
14         A a = A(addr);
15         ua = a.foo("call foo directly");
16     }
17 }
```

创建一个foo函数.

创建一个callAFooDirectly函数.

2. 使用address类型的call()函数调用.

```
contract C {
    function callAFooByCall(address addr) public returns (bool){
        bytes4 funcsig = bytes4(keccak256("foo(string)"));
        if (addr.call(funcsig,"call foo by func call"))
            return true;
        return false;
    }
}
```

3. 代理调用 delegatecall()

fallback() 函数

```
function() public [payable]{
  ......
}
```

- 匿名函数，没有参数也没有返回值。
- 在两种情况下会被调用：
  - 直接向一个合约地址转账而不加任何data
  - 被调用的函数不存在
- 如果转账金额不是0，同样需要声明payable，否则会抛出异常。

智能合约的创建与运行

创建合约：外部帐户发起一个转账交易到0x0的地址
- 转账的金额是0，但是要支付汽油费
- 合约的代码放在data域里

智能合约运行在EVM（Ethereum Virtual Machine）上 → 256位，uint也是256位.

汽油费 gas fee

智能合约是个Turing-complete Programming Model
- 出现死循环怎么办？全节点收到一个对智能合约的调用，怎么知道这个调用执行起来会不会是一个死循环？ → 没有办法（halting problem，不可解）

因此，ETH引入了汽油费机制.

全节点收到了对智能合约的调用，计算最多需要的gas fee并从账户中扣除相应的ETH。待调用完成后，得出实际需要的gas fee，多退少回滚.

EVM中不同指令消耗的汽油费是不一样的
- 简单的指令很便宜，复杂的或者需要存储状态的指令就很贵

简单的指令 → 四则运算；复杂的或者需要存储状态 → 如取哈希.

ETH交易的错误处理.

ETH交易具有原子性，即一个交易要么不执行，要么全部执行，不会执行一部分.

可以抛出错误的语句：
- assert(bool condition):如果条件不满足就抛出—用于内部错误。
- require(bool condition):如果条件不满足就抛掉—用于输入或者外部组件引起的错误。

```
function bid() public payable {
    // 对于能接收以太币的函数，关键字 payable 是必须的。

    // 拍卖尚未结束
    require(now <= auctionEnd);
```

- revert():终止运行并回滚状态变动。→ 无条件抛错.

问题：嵌套调用发生错误是否会连锁式回滚

答：有些会有些不会 { 1. 直接调用：✓ 2. call() 调用：×

➢ 一个合约直接向一个合约帐户里转账，没有指明调用哪个函数，仍然会引起嵌套调用

Block header

```
GasLimit    uint64    `json:"gasLimit"    gencodec:"required"`
GasUsed     uint64    `json:"gasUsed"     gencodec:"required"`
```

→ 该区块能够消耗资源的上限

→ 每次可增加（或下降）1/1024，这种机制使得系统的 gas limit 反映全体矿工的平均意见

问题：一个全节点打包一些交易到区块里，这些交易中有一些是对智能合约的调用，那么这个全节点应先执行智能合约再挖矿，还是先挖矿再执行智能合约？

汽油费的扣除机制

① ETH 的3棵树（状态树、交易树、收据树）都是全节点在本地维护的数据结构

② 全节点收到对智能合约的调用的时候，在本地将账户余额扣除

③ 然后取得记账权的节点发布区块，将它本地维护的3棵树上传

ETH 挖矿过程

① 全节点打包交易、执行对智能合约的调用，调整智能合约的内容。

② 求得3棵树的根哈希值。

③ 试 nonce，取得记账权、发布区块

④ 别的没有取得记账权的区块，要独立验证新发布区块及其包含的 tx 和智能合约的合法性、

综上，应当先执行，再挖矿

接上问：如果先执行后，没有挖到矿，是否有补偿？

答：没有任何补偿。并且还要验证别的出块节点出块内交易及合约执行的合法性。

再接上问：既然如此，是否会有区块行政不作为（即不验证新发区块合法性）。这种行为有何后果？如何防范这种行为？

答：如果某节点跳过验证步骤，以后就无法挖矿了。因为不验证即不执行交易，即无法更新本地的三棵树，即无法再挖矿

注：发布的区块块头中，没有状态树的内容（因为太多），只有其根hash值。

Receipt 数据结构

```
45  // Receipt represents the results of a transaction.
46  type Receipt struct {
47          // Consensus fields
48          PostState         []byte `json:"root"`
49          Status            uint64 `json:"status"`
50          CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
51          Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
52          Logs              []*Log `json:"logs"              gencodec:"required"`
53
54          // Implementation fields (don't reorder!)
55          TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
56          ContractAddress common.Address `json:"contractAddress"`
57          GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
58  }
```

（Status：交易执行情况如何）

问题：智能合约是否支持多线程（多核并行）处理

答：不支持。状态机必须是完全确定。而多线程如果对内存访问顺序不同，造成的结果可能也会不同。

除了多线程，其它包括"产生随机数"

智能合约可以获得的区块信息

- `block.blockhash(uint blockNumber) returns (bytes32)`：给定区块的哈希—仅对最近的 256 个区块有效而不包括当前区块
- `block.coinbase` (`address`)：挖出当前区块的矿工地址
- `block.difficulty` (`uint`)：当前区块难度
- `block.gaslimit` (`uint`)：当前区块 gas 限额
- `block.number` (`uint`)：当前区块号
- `block.timestamp` (`uint`)：自 unix epoch 起始当前区块以秒计的时间戳

智能合约可以获得的调用信息

- `msg.data` (`bytes`)：完整的 calldata
- `msg.gas` (`uint`)：剩余 gas
- `msg.sender` (`address`)：消息发送者（当前调用）
- `msg.sig` (`bytes4`)：calldata 的前 4 字节（也就是函数标识符）
- `msg.value` (`uint`)：随消息发送的 wei 的数量
- `now` (`uint`)：目前区块时间戳（`block.timestamp`）
- `tx.gasprice` (`uint`)：交易的 gas 价格
- `tx.origin` (`address`)：交易发起者（完全的调用链）

地址类型

这里的<address>都是被动方.

`<address>.balance` (`uint256`):
以 Wei 为单位的 地址类型 的余额。

`<address>.transfer(uint256 amount)` :
向 地址类型 发送数量为 amount 的 Wei，失败时抛出异常，发送 2300 gas 的矿工费，不可调节。

`<address>.send(uint256 amount) returns (bool)` :
向 地址类型 发送数量为 amount 的 Wei，失败时返回 `false` ，发送 2300 gas 的矿工费用，不可调节。

`<address>.call(...) returns (bool)` :
发出底层 `CALL` ，失败时返回 `false` ，发送所有可用 gas，不可调节。

`<address>.callcode(...) returns (bool)` :
发出底层 `CALLCODE` ，失败时返回 `false` ，发送所有可用 gas，不可调节。

`<address>.delegatecall(...) returns (bool)` :
发出底层 `DELEGATECALL` ，失败时返回 `false` ，发送所有可用 gas，不可调节。

三种发送ETH的方式

```
<address>.transfer(uint256 amount)
<address>.send(uint256 amount) returns (bool)
<address>.call.value(uint256 amount)()
```

从一个例子开始：简单拍卖

(不允许中途退出，可重复出价，重复出价补差额即

```
1  pragma solidity ^0.4.21;
2
3  contract SimpleAuctionV1 {
4      address public beneficiary;      //拍卖受益人
5      uint public auctionEnd;          //结束时间
6      address public highestBidder;    //当前的最高出价人
7      mapping(address => uint) bids;   //所有竞拍者的出价
8      address[] bidders;               //所有竞拍者
9      bool ended;                      //拍卖结束后设为true
10
11     // 需要记录的事件
12     event HighestBidIncreased(address bidder, uint amount);
13     event AuctionEnded(address winner, uint amount);
14
15     /// 以受益者地址 `_beneficiary` 的名义，
16     /// 创建一个简单的拍卖，拍卖时间为 `_biddingTime` 秒。
17     constructor(uint _biddingTime,address _beneficiary) public {
18         beneficiary = _beneficiary;
19         auctionEnd = now + _biddingTime;
20     }
```

```
/// 对拍卖进行出价
/// 随交易一起发送的ether与之前已经发送的ether的和为本次出价
function bid() public payable {
    // 对于能接收以太币的函数，关键字 payable 是必须的。

    // 拍卖尚未结束
    require(now <= auctionEnd);
    // 如果出价不够高，本次出价无效，直接报错返回
    require(bids[msg.sender]+msg.value > bids[highestBidder]);

    //如果此人之前未出价，则加入到竞拍者列表中
    if (!(bids[msg.sender] == uint(0))) {
        bidders.push(msg.sender);
    }
    //本次出价比当前最高价高，取代之
    highestBidder = msg.sender;
    bids[msg.sender] += msg.value;
    emit HighestBidIncreased(msg.sender, bids[msg.sender]);
}
```

```
/// 结束拍卖，把最高的出价发送给受益人，
/// 并把未中标的出价者的钱返还
function auctionEnd() public {
    //拍卖已截止
    require(now > auctionEnd);
    //该函数未被调用过
    require(!ended);

    //把最高的出价发送给受益人
    beneficiary.transfer(bids[highestBidder]);
    for (uint i = 0; i<bidders.length;i++){
        address bidder = bidders[i];
        if (bidder == highestBidder) continue;
        bidder.transfer(bids[bidder]);
    }

    ended = true;
    emit AuctionEnded(highestBidder, bids[highestBidder]);
}
```

退钱

智能合约、code is law. 如果出bug是无法修复的。不存在后门或者超级管理者

→ 与去中心化精神而驰

后部有部分内容理解有难度，需要 solidity 再深入研究



区块链技术与应用

第23讲：The DAO = Decentralized Autonomous Organization

DAC: Decentralized Autonomous corporation.

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

the DAO: 某个组织，本质：运行在ETH的智能合约

Split DAO → Child DAO 取回钱的方法

DAO是去中心化自治组织。其目的是为组织规则以及决策机构编写代码，从而消除书面文件的需要，以及减少管理人员，从而创建出一个去中心化管理架构。

下面是其运作方式：

> · 一组人来编写运行组织的智能合约（程序）
>
> · 有初始融资阶段，在这一阶段人们添加资金来购买代币，来代表其所有权——这个过程叫做众销，或者首次代币发行（ICO）——为其提供所需资源。
>
> · 当融资阶段结束后，DAO就开始运行。
>
> · 之后人们就开始向DAO谏言献策该如何使用这笔钱，购买DAO的成员就有资格对这些提案进行投票。

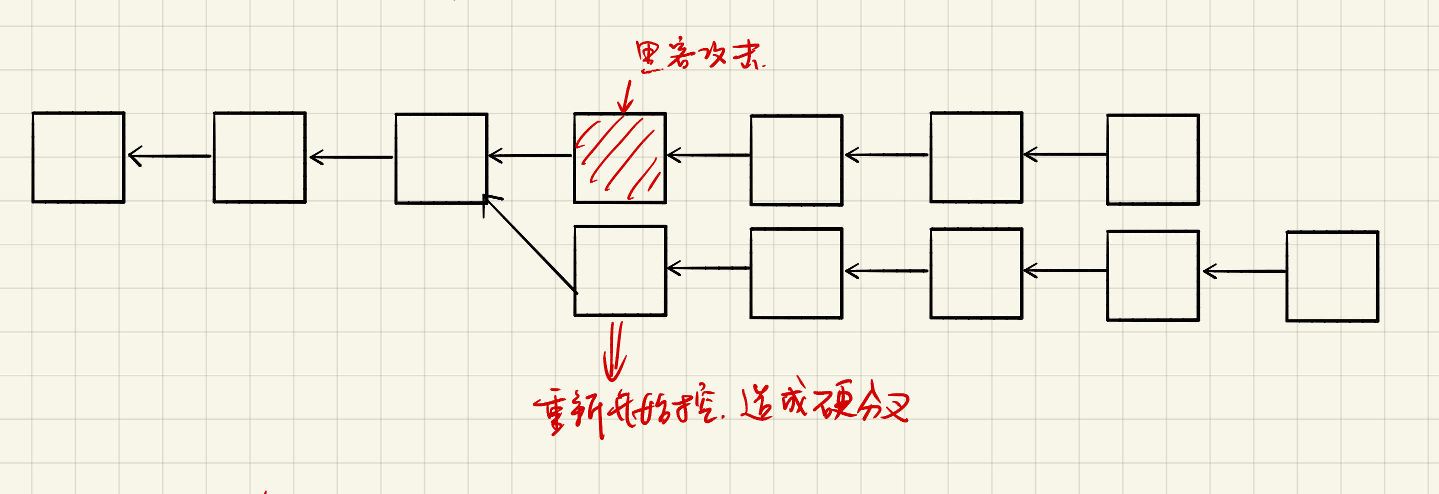
The DAO当时拥有ETH世界拥15%的以太币，如果黑客攻击成功，会拥有约1/3的The DAO的ETH.

The DAO对于ETH来说，已经是 too big to fall 了.

如何解决？

方案A：在黑客攻击之前的区块进行分叉。

回答：不可行，因为这样不仅黑客的交易发生回滚，其它合法交易和智能合约也会回滚。



必须要精确定位黑客交易才可以

ETH方案：2步走 { ① 锁定黑客账户 / ② 清退黑客账户的钱 }

① 锁定黑客账户：升级软件，所有与 the DAO 有关的账户不能再发生交易（软分叉）

遗憾的是，升级后的软件有bug（与the DAO有关账户发生交易时，被deny，但不收取gas fee），导致发生 denied tx attack，于是很多矿工不得以回滚软件版本

② 清退黑客账户的钱：软分叉方案失败后，ETH进行了硬分叉，发布了软件，将the DAO账户里的钱转移到一个新的智能合约，这个合约只有一个功能：退钱（withdraw）

硬分叉方案引起了 ETH community 的剧烈讨论，大家开始用币投票（2个智能合约，投票后锁币），最后大多数人赞成硬分叉方案（事实上很多人没有参与投票）。

很多矿工仍留在旧链上挖矿，有交易所上市了旧链的以太币，改为ETC
新链仍叫 ETH.



区块链技术与应用

第24讲：反思

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

Is smart contract really smart?

- smart contract is anything but smart.

Irrevocability is a double edged sword.

Nothing is Irrevocable.

实际上，什么都可以改，不要迷信不可篡改

Is solidity the right programming language?

BTC：简单脚本语言，很多复杂过程无法描述

ETH：图灵完备，简单来讲，一切可计算的问题都能计算，这样的虚拟机或者编程语言就叫图灵完备的。

适中：能够实现智能合约的功能，但又不容易像ETH出现安全漏洞。
介于BTC与ETH之间

未来智能合约可能的发展方向：出现常用条款模板，出现专门编写合约的机构

Many eyeball fallacy

开源代码，理论上开源了会有很多人去检查，会更加安全。但实际上真正去看的人很少，即使看也不一定有专业知识去发现bug。

What does decentralization mean?

最后硬分叉成功的原因，并不是ETH开发团队的代码，而是大多数矿工用挖矿行动的支持。

去中心化≠全自动化，≠让机器决定一切，不能有人为的干预

去中心化≠已经制定的规则不能修改，而是说对规则的修改，要用去中心化的方式进行

decentralized ≠ distributed

1个去中心化的系统，必然是分布式的

但是分布式系统 ≠ 去中心化

state machine 的目的不是为了加速，而是为了容错.

mission critical application.

- air traffic control
- stock exchange
- space shuttle



区块链技术与应用

第25讲：美链 Beauty chain

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

代币的转账不同于ETH的转账，代币的转账本质是调用智能合约进行转账操作。

美链 → BEC → ICO

- 美链(Beauty Chain)是一个部署在以太坊上的智能合约，有自己的代币BEC。
  - 没有自己的区块链，代币的发行、转账都是通过调用智能合约中的函数来完成的
  - 可以自己定义发行规则，每个账户有多少代币也是保存在智能合约的状态变量里
  - ERC 20是以太坊上发行代币的一个标准，规范了所有发行代币的合约应该实现的功能和遵循的接口
  - 美链中有一个叫batchTransfer的函数，它的功能是向多个接收者发送代币，然后把这些代币从调用者的帐户上扣除

ERC = Ethereum Request for comments.

某机构A欲ICO发行代币，A只需在ETH中创建一个智能合约。某B欲购买其代币，只需向该智能合约转入ETH，该智能合约就会为B在智能合约下开设一个子账户，并存入相应数量的代币

batchTransfer的实现

```
function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]].add(_value);
        Transfer(msg.sender, _receivers[i], _value);
    }
    return true;
  }
}
```

如果value很大，会溢出，则减很小的数目。相当于系统中凭空发行很大数目BEC；

(_value) 加还是照加，减会减溢出后的值。



北京大学
PEKING UNIVERSITY

区块链技术与应用

第26讲：课程总结

主讲老师：肖臻 研究员
课程资料：http://zhenxiao.com
新浪微博：北大肖臻

错误应用场景1：用BTC转账技术加速保险理赔.

解释：保险理赔慢，并不是因为支付慢，而是因为保险事项需要人工确认.

错误应用场景2：防伪追溯，如蔬菜从产地到市场上链.

解释：如果本身写入区块链的就是假的内容或过程中实物被掉包，这些都是区块链无法解决的问题

加密货币不应该被用作同传统金融体系竞争的方面，而应该去补足传统金融体系无法解决的问题.

e.g. 餐馆、电商 没有必要去接受BTC，因为支付宝已经很好解决了线下付款问题.

现有金融体系，缺乏一种方便的跨国转账方式，并且这种转账方式，要同信息传播融合在一起.

下一代互联网是"价值交换网络"，而我们目前只是"信息交换网络"

Information can flow freely on the Internet, but payment cannot.

对于智能合约与传统合同的看法.

程序化是大趋势：Software is eating the world.

Democrazy is the worst form of Government except for all those other forms that have been tried from time to time. 丘吉尔

Is decentralization always a good thing?

Of course not.